

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 08-255104

(43)Date of publication of application : 01.10.1996

(51)Int.Cl.

G06F 12/00
G06F 12/00
G06F 9/06

(21)Application number : 07-325290

(71)Applicant : AT & T CORP

(22)Date of filing : 14.12.1995

(72)Inventor : KORN DAVID GERARD
KIEM-PHONG VO

(30)Priority

Priority number : 94 355889 Priority date : 14.12.1994 Priority country : US

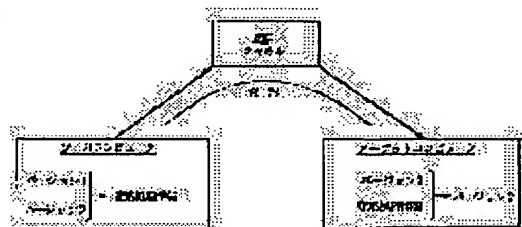
(54) EFFICIENT AND HIGHLY SAFE UPDATING OF SOFTWARE AND DATA

(57)Abstract:

PROBLEM TO BE SOLVED: To provide a method for reducing a transmission data amount, preventing interception and updating a software or data from a remote location.

SOLUTION: Relating to a device for updating a data file from an old version to a new version, the old version and the new version are compared and deformation processing information including information relating to similarity and difference is led out. Further, in order to lead out the new version, without referring to the new version itself, the deformation processing information is used and the old version is processed. Even without transmitting an updated file itself, by transmitting the deformation processing information to plural places, the plural versions of an original file such as the records of banks present at the plural places are updated.

Generally, since the entire contents of the new version are not included in the deformation processing information, this method is provided with high resistance to the interception.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision
of rejection]

[Date of requesting appeal against examiner's
decision of rejection]

[Date of extinction of right]

Copyright (C); 1998,2003 Japan Patent Office

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平8-255104

(43) 公開日 平成8年(1996)10月1日

(51) Int.Cl. ⁵	識別記号	庁内整理番号	F I	技術表示箇所
G 0 6 F 12/00	5 1 7	7623-5B	G 0 6 F 12/00	5 1 7
	5 1 0	7623-5B		5 1 0 B
9/06	4 1 0		9/06	4 1 0 P

審査請求 未請求 請求項の数 8 O L (全 38 頁)

(21) 出願番号 特願平7-325290

(22) 出願日 平成7年(1995)12月14日

(31) 優先権主張番号 08/355889

(32) 優先日 1994年12月14日

(33) 優先権主張国 米国 (U S)

(71) 出願人 390035493

エイ・ティ・アンド・ティ・コーポレーション

AT&T CORP.

アメリカ合衆国 10013-2412 ニューヨーク
ニューヨーク アヴェニュー オブ
ジ アメリカズ 32

(72) 発明者 ディヴィッド ジェラルド コーン

アメリカ合衆国 10003 ニューヨーク,
ニューヨーク, エー107, マーサー スト
リート 303

(74) 代理人 弁理士 岡部 正夫 (外2名)

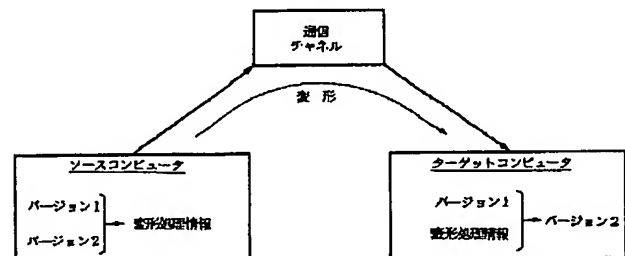
最終頁に続く

(54) 【発明の名称】 ソフトウェアおよびデータの効率的かつ安全性の高い更新

(57) 【要約】

【課題】 伝送データ量を削減し、かつかつ傍受されにくい、遠隔地からソフトウェア又はデータを更新する方法を提供する。

【解決手段】 本発明は、旧バージョンから新規バージョンへデータファイルを更新する装置に関する。本発明は、旧バージョンと新規バージョンを比較し、類似と相違に関する情報を含む変形処理情報を導き出す。さらに、本発明は、新規バージョンを導き出すために、新規バージョン自体を参照せずに、変形処理情報を用いて、旧バージョンの処理を行う。本発明を用いると、更新されたファイル自体を送信しなくても、変形処理情報を複数の場所に送信することにより、複数の場所にある銀行の記録などのような元のファイルの複数バージョンを更新することができる。一般に、変形処理情報には新規バージョンの内容全体が含まれていないことから、この方法は傍受に対して高い抗力を持つ。



【特許請求の範囲】

【請求項 1】 コンピュータにおいて、

a) 第二バージョンにアクセスしなくても、第一バージョンに基づいてバイナリファイルの第二バージョンの復元を可能にする命令を作成するプログラム手段から成る改良。

【請求項 2】 a) 連結されたときに第二バージョンを復元する文字列を各々が作成する一連の命令を生成する段階と、

b) 各命令を実行する段階とから成る、第一バージョンから導き出されたバイナリファイルの第二バージョンを復元する方法。

【請求項 3】 デジタルコンピュータの場合、

a)

i) 第一および第二ファイルを検査し、

ii) 第二ファイルを復元するため、実行されたときに、

A) 第一ファイルの一部と、

B) 第二ファイルへのアクセスを行わずに第二ファイルの一部とを結合する一連の命令を作成する第一プログラム手段から成る改良。

【請求項 4】 第二ファイルの復元を行うために一連の命令を実行する第二プログラム手段を具備する請求項 3 に記載の改良。

【請求項 5】 a) 新規バージョンと旧バージョンとを比較し、かつ、

i) 新規バージョンが旧バージョンと似ている類似語句と、

ii) 新規バージョンが旧バージョンと異なる相違語句を識別し、

b) 旧バージョンに各類似語句が出現しているアドレスを記憶し、

c) 新規バージョンに出現しているアドレスと共に各相違語句を記憶する段階とから成る、コンピュータファイルの新規バージョンを退避する方法。

【請求項 6】 a) 第一サイトにおいて、第一ファイルと第二ファイルの

i) 類似と、

ii) 相違

を検出する段階と、

b) 第二サイトにおいて、第一ファイルのコピーを保持する段階と、

c)

i) 類似の位置と、

ii) 相違自体と、

iii) (c) (i)、(c) (ii)、および第二ファイルに基づく第二ファイルの復元を可能にする情報を、第二サイトに送信する段階とから成る情報の復元法。

【請求項 7】 a)

i) 新規バージョンの指定位置に旧バージョンの指定部分をコピーすることと、

ii) 新規バージョンの指定位置に新規バージョンの指定部分をコピーすることと、

iii) 新規バージョンの指定位置に指定バイトを加えることのうち、1つまたはそれ以上を行うよう指示する命令を受信する段階と、

b) 命令を実行する段階とから成る、ファイルの旧バージョンを新規バージョンへ更新する方法。

【請求項 8】 a)

i)

A) 第一バージョンまたは第二バージョンの発信元位置から、

B) 第二バージョンの宛先位置へ、文字がコピーされるよう指示する COPY 命令と、

ii) 指定文字が第二バージョンの指定位置に加えられるよう指示する ADD 命令とから成る、並びを導き出す手段から成るファイルの第一バージョンと第二バージョンの相違を表すデータを作成するシステム。

【発明の詳細な説明】

【0001】

【産業上の利用分野】 本発明により用いられるソースコードを記載した付録を添付する。本発明は、伝送されるデータ量を削減し、かつ傍受されにくい、遠隔地からソフトウェアまたはデータもしくは両者を更新する方法に関する。

【0002】

【従来の技術】 電信電話会社の電話システムを介したコンピュータ間の電子データ通信には、スピードと機密保護が必要とされている。伝送前にデータを圧縮することにより伝送速度を向上できる一方で、暗号化により機密保護が可能である。データ圧縮は、データ源における冗長度を低下させ、かつ暗号解読への抗力を高めることから、暗号化にも有効な働きをする。しかし、データファイルが圧縮ならびに暗号化された場合、圧縮および暗号化されたデータは、単一のデータ源に基づくものとなる。単一のデータ源に基づくデータ圧縮および暗号化は、完全なものとはいえない。データの冗長度が高くなければ、圧縮はうまく機能せず、最新の暗号解読法と共に機能する高速コンピュータは、大抵の暗号化データを解読することができるからである。多くの場合、最初のデータ源に関する複数の新規バージョンが、データに若干の変更が加えられた後に頻繁に伝送される。この場合の変更が若干であるとは、全バージョンが本質的に類似していることを意味している。本質的に類似していれば、その類似性を利用して、あるバージョンを別のバージョンに変形する最小の変形処理情報を計算することが可能である。

【0003】 このような変形処理情報の計算方法は、データディファレンシング（データ間の差の計算）と呼ば

れている。データディファレンシングは伝達されるデータ量を削減する働きをすることから、これにより伝送速度が向上する。さらに、旧バージョンを持たない傍受者は、伝送されたデータからあまり多くの情報を引き出すことができないため、データディファレンシングは、プライバシーの保護にも役立つ。これまで数多くの圧縮およびディファレンシング技術が、検討されてきた。「情報理論に関するIEEEトランザクション (IEEE Transactions on Information Theory)」1977年5月号の23 (3) (337頁～343頁)に記載されたJ. ZivならびにA. Lempelによる「順次データ圧縮のための普遍アルゴリズム (A Universal Algorithm for Sequential Data Compression)」の記事では、単一のデータファイルを圧縮する技術について述べられている。この技術は、データの順序を分析し、かつ、可能であれば、各位置においてすでに分析された部分にある別のセグメントと一致する最長セグメントを識別することによって機能する。このようなセグメントが検出された場合、一致した位置と一致した長さを用いて暗号化が行われる。また、一致しなかったデータは、そのまま出力される。

【0004】このアルゴリズムの具体化が、1984年8月7日にW. L. Eastman, A. Lempel, およびJ. Ziv. に付与された米国特許4, 464, 650の主題であった。このEastman-Lempel-Ziv圧縮方法では、圧縮解除中も圧縮中とほぼ同じ作業量をこなさなければならないことから、圧縮解除の際に低速となる。さらに、この方法は、データディファレンシングに応用できない。UNIXシステムでは、第一ファイルから第二ファイルへの変形の際に削除または追加する必要のある一連の行を作成する「diff」プログラムを用いて、(バイナリ以外の)テキストファイル間の違いを算出することができる。「diff」による方法を採用した場合、テキスト行の若干の変更によって、極めて大幅な変形は別のものとみなされることから、そうした大幅な変形を行う可能性も出てくる。さらに、この方法は、テキストファイルでしか機能しないことから、応用範囲が限られている。「コンピュータシステムに関するACMトランザクション (ACM Transactions on Computer Systems)」1984年11月号の2 (4) (309頁～321頁)に記載されたWalter F. Tichyによる「ブロックの移動によるストリングからストリングへの訂正上の問題 (The String-to-String Correction Problem with Block Moves)」の記事では、データファイルのあるバージョンから別のバージョンへ変形を行うブロック移動と呼ばれる一連の命令を計算するアルゴリズムについて述べられて

いる。ブロック移動 (block-move) は、第一バージョンの別のセグメントと一致する第二バージョンのデータセグメントである。このアルゴリズムは、本質的に実現されないものであり、第二バージョンに固有の冗長性がある場合に、それを活用して第一バージョンから第二バージョンを作成するのに必要な変形を最小限度に抑えることができない。

【0005】

【発明の概要】本発明の一態様において、コンピュータプログラムにより、ファイルの第一バージョンと第二バージョンの新規バージョンを比較し、一連の命令を作成する。この命令により、第一バージョンに基づいた第二バージョンの復元が可能になる。上記の命令は、2つのタイプに分けられる。ひとつはCOPY命令であり、指定された一連の文字を構成中のファイルにコピーするよう命令する。指定された文字列は、第一バージョンまたは構成中のファイルのいずれにも存在可能である。もうひとつは、ADD命令であり、このADD命令に伴う一連の文字の追加を指示する。この2種類の命令が連続的に実行されると、コピーならびに追加された各文字列の連結が行われ、第二バージョンの復元が行われる。

【0006】

【実施例】以下に、簡単な類比により、本発明のより基本的な側面を一部説明する。また、本発明の一部を実現するコンピュータコードについての技術的な説明は、「本発明について」と題した項目にて行う。

類推

ある新聞記者と新聞編集者が、記事を一緒に、しかし、異なった場所で書くものとする。さらに、両者が図1Aに示すバージョン1を書いたと仮定する。また、この新聞記者が図1Bに示す変更を行った結果、図1C (四角枠は変更した位置を示している) に示すようなバージョン2となり、このバージョンを編集者に送信したいと考えていると仮定する。そこで、次の手順を用いれば、バージョン2の内容全体を送信せずに、編集者にバージョン2の送信が可能である。

【0007】手順

まず初めに、記者は、図1Aに示すように、バージョン1の各語に位置を割り当てる。50語に対し、50箇所の位置がある。次に、記者は、バージョン1の単語表を確認する。単語表は、図1Aに示すような使用される語のリストすなわち一覧表である。ただし、各語は、単語表に一度だけしか現れない。例えば、位置18の語「them」は単語表に記入されない。この語がすでに位置16にあるからである。各語が一度しか記入されていないことを確認するために、記者は、各単語を入力する度に、その語が表にあるかどうか見ながらチェックする。例えば、記者が、最初の単語「When」を入力する。次に、位置2の語「buying」を取り上げ、単語表の最初の単語、すなわち、「When」と照合する。一

致しなければ、記者は、「buying」を単語表の位置 2 に記入する。記者は、「them」という語が位置 1 8 にくるまでこのようにして処理を進める。記者がこの語を単語表内にあるそれまでに入力された 1 7 の項目と照合してみると、「them」が単語表の位置 1 6 にある「them」と一致することに気がつく。したがって、後の「them」は、前の「them」と重複するため単語表に記入されない。

【0008】バージョン 1 の最後の語について一致の確認が行われると、単語表は完了する。バージョン 1 には、全体の長さに対し 5 0 語含まれているが、単語表の長さが示す通り、異なる語は 3 8 語しかない。次に、記者は、図 1 C にあるように、バージョン 2 の単語表を作成する。バージョン 2 には、新語として示されている追加の 3 語を除き、バージョン 1 と同じ単語表がある。これで、記者は、編集者にバージョン 2 を送信する準備が整ったことになる。記者は、次の 6 つのメッセージすなわち命令を送信してバージョン 2 を伝送する。各命令は、図 1 D ~ 図 1 I の各図によって理解できる。

1. COPY 2 1
2. ADD 1 Hawaiian
3. COPY 3 3
4. ADD 1 all
5. COPY 37 8
6. ADD 2 a brick

【0009】命令 1

事前配列により、記者と編集者は、図 1 D に示すように、位置 1 にポインタを設定する。最初の命令「COPY 2 1」は、「バージョン 1 の位置 1 から始まる 2 語の並びをコピーせよ」を意味している。（各命令では、該当する語がバージョン 2 のポインタの位置に配置されることが暗黙の了解となっている。）この動作が、図 1 D に示されている。ただし、構文は、以下の通りである。

COPY [語数、バージョン 1 の開始位置]

命令を実行した後、編集者は、COPY 命令にある「語数」（この場合、2 語）分だけポインタを移動させる。これにより、ポインタは、図 1 E に示す位置に設定される。

【0010】命令 2

命令 2 では「ADD 1 Hawaiian」とあり、「Hawaiian」という 1 語を付け加えよ」を意味している。この動作は、図 1 E に示されている。ただし、構文は以下の通りである。

ADD [語数、該当する語]

命令を実行した後、編集者は、ADD 命令にある「語数」（この場合、1 語）分だけポインタを移動させる。これにより、ポインタは、図 1 F に示す位置に設定される。

【0011】命令 3

命令 3 の「COPY 3 3」は、「バージョン 1 の位置 3 から始まる 3 語の並びをコピーせよ」を意味している。この動作は、図 1 F に示されている。ただし、構文は命令 1 と同一である。命令を実行した後、編集者は、COPY 命令にある「語数」（この場合、3 語）分だけポインタを移動させる。これにより、ポインタは、図 1 G に示す位置に設定される。

【0012】命令 4

命令 4 では「ADD 1 all」とあり、「all」という 1 語を付け加えよ」を意味している。この動作は、図 1 G に示されている。命令を実行した後、編集者は、ADD 命令にある「語数」（この場合、1 語）分だけポインタを移動させる。これにより、ポインタは、図 1 H に示す位置に設定される。

【0013】命令 5

命令 5 の「COPY 37 8」は、「バージョン 1 の位置 8 から始まる 37 語の並びをコピーせよ」を意味している。この動作は、図 1 H に示されている。ただし、構文は命令 1 と同一である。命令を実行した後、編集者は、COPY 命令にある「語数」（この場合、37 語）分だけポインタを移動させる。これにより、ポインタは、図 1 I に示す位置に設定される。

【0014】命令 6

命令 6 では「ADD 2 a brick」とあり、「a brick」という 2 語を付け加えよ」を意味している。この動作は、図 1 I に示されている。これで、バージョン 2 が作成されたことになる。

【0015】重要な特徴

図 1 C に示されるバージョン 2 には、43 語が含まれており、語自体は、163 文字から構成されていた。しかし、6 つの命令には 27 文字プラス命令自体（COPY と ADD）が含まれていた。各命令と各文字を 1 バイトとしてコード化した場合、全体のメッセージは、27 + 6 文字、すなわち、33 文字となる。33 文字を送信すれば、163 文字を送信した場合に比べて大幅に時間が短縮される。

【0016】本発明について

上記の類比の説明は簡単に述べたものであり、本発明の特徴のすべてを示しているわけではない。図 2 では、ファイルに関する 2 つのバージョンの他に、バージョン 1 と組み合わせてバージョン 2 を作成できるようにする「変形処理情報」を示している。図 3 の手続きでは、復元が行われる。この手続きについてこれから説明する。まず初めに、一般的なパターンが綿密に作成される。予備知識として、重要点を 4 つ認識しておく必要がある。ひとつは、各バージョンの各文字は、図 2 A に示す通り、番号付けされた位置を占めている。例えば、バージョン 1 の場合、最初の「a」は位置 0 を占有している。最初の「b」は位置 1 を占有している等である。第二に、バージョン 2 の番号付けされた位置は、バージョン

1の番号付けされた位置の中で最も高い数字の位置の後ろから開始する。したがって、バージョン1の最も数字の高い位置が「15」であることから、バージョン2は「16」から開始することになる。第三に、ポインタ（図3の手続き中にある変数）は、図2Aに示すような現在位置を示している。第四に、「変形処理情報」は、複数の命令により構成されている。命令には2種類、すなわちADDとCOPYがある。次に、この2種類の命令の動作について、バージョン1からバージョン2がどのように構成されるのかを示しながら説明したい。

【0017】命令1

図2Bでは、「変形処理情報」の命令1が実行されている。命令「COPY 4 0」は、基本的に、「バージョン1の位置0から始まる4文字をバージョン2のポインタの位置にコピーせよ」を示している。（下線の語は、命令内の語を示している。）この動作は、図2Bに示されている。上記の命令では、バージョン2の作成に使用される文字を、命令自体からではなく、バージョン1から取得している。このため、電話送信により命令を取得する場合でも、傍受者は、バージョン1にアクセスできないと推定されるため、バージョン2の作成に使用できる情報は一切得ることができない。構文は、以下の通りである。

COPY [語数、開始位置]

【0018】命令2

図2Cでは、命令2が実行される。命令「ADD 2 x, y」は、事実上、「xとyの2文字をポインタの位置を起点にバージョン2に追加せよ」を意味している。この命令では、バージョン2の作成に用いられる文字を命令自体から得ている。その結果、傍受者は、バージョン2に関する情報の一部を取得できる。しかし、実際には、この種の命令は、非常に多種多様な（情報を一切含まない）COPY命令と混合して用いられることが予想されるため、傍受者は、バージョン2について重要な情報を得ることはできない。にもかかわらず、理論的には、ADD命令のみが「変形処理情報」に含まれる場合もあり得ることであり、その場合、傍受者は、バージョン2をそのまま取得することになる。あるいは、ADD命令の数が膨大になることもある。傍受者がこうしたADD命令から情報を取得できないようにするために、暗号化オプションが提供されており、内容については後に述べる。ADD命令の構文は以下の通りである。

ADD [語数、該当文字]

【0019】命令3

図2Dでは、命令3が実行されている。命令「COPY 6 20」は、事実上、次の内容を示している。位置20とポインタとの間にある文字を用いて、長さが6文字のコピーを作成し、ポインタの位置にコピーした文字を配置せよ。命令を実行した結果、図示されるように、x y x y x yとなる。（別の例として、COPY文で

「COPY 6 19」としてあれば、事実上、次の内容を意味している。位置19（20ではない）とポインタとの間にある文字を用いて、長さが6文字のコピーを作成し、ポインタの位置にコピーした文字を配置せよ。この場合、位置19とポインタとの間にある文字列はd x yであることから、ポインタの位置に付け加えられた文字の並びはd x y d x yとなる。）

この命令の場合、バージョン2の作成に用いられる文字がバージョン2から取得される点が重要である。したがって、COPY命令は指定されたアドレスによって2つのデータ源を使用していることがわかる。この命令では、アドレス（すなわち、「COPY 6 20」の「20」）が、バージョン2を示している（すなわち、アドレスは15を上回る）。したがって、バージョン2が、このコマンドのデータ源である。逆に、アドレスが15以下であれば、バージョン1がデータ源として用いられるはずである。この方法を用いると、事実上、ADD命令を使用して単語表を拡張することができる。したがって、（a）バージョン1の中になく、（b）拡張された単語表にある（以前に追加された）文字がバージョン2に含まれていることが明らかになった場合、COPY命令が使用できる。したがって、この場合のCOPY命令には2つの利点がある。ひとつは、COPY命令では、文字の総数と開始位置を示すだけで大量の文字を挿入することができる。逆に、ADD命令を用いた場合、送信される文字自体を必要とすることから、より長い送信が必要となる。もうひとつは、上記の通り、命令を傍受した者がバージョン1を取得するようなことがない限り、COPY命令には一切の情報が含まれていないことになる。

命令4

図2Eでは、命令4が実行されている。命令「COPY 5 9」は、事実上、「バージョン1の位置9から始まる5文字をコピーせよ」を示している。この命令は、命令1とよく似ている。

【0020】図3のプロセスに関する参照事項

図3は、コンピュータがバージョン1と「変形処理情報」との組み合わせからバージョン2を作成するプロセスを示している。図3では、2行目において、図2B～図2Eに示されているポインタの計算に用いられる変数cの初期化が行われている。ポインタの計算は、9行目、13行目、および14行目で適宜行われている。

命令1

図2に示される「変形処理情報」の命令1（COPY 4 0）は、図3の13行目で実行されている。変数pは、原始データの開始位置を示しており、命令の中から得ることができ、この場合、0である。pはn（12行目）よりも小さいことから、データ源はバージョン1にある。その結果、1F文により13行目が実行されると、バージョン1からデータがコピーされる。このと

き、バージョン1+pの位置、すなわち位置0から開始する。変数sは、文字数を表しており、7行目の命令から得ることができ、この場合、4である。ポインタは、16行目において更新される。

命令2

命令2 (ADD 2 x, y) は、8行目のIF文により、9行目で実行される。これにより、バージョン2+cの位置にある長さs (2に等しい) の文字列が設定される。ポインタは、16行目で更新される。

命令3

命令3 (COPY 6 20) は、12行目[p (命令から得られ、20に等しい) はnより小さい]により14行目において実行される。このコピー関数は、データ源としてバージョン2を用い、バージョン2+cから開始する。コピー関数は、文字数sの長さを有する並びである。(変数sは[ポインタ-20]により得られ、20の数字は命令から得られる。)

命令4

命令4は、命令1のように13行目において実行される。

【0021】「変形処理情報」の作成

図5は、各バージョンを1文字ずつ確認し、かつバージョン1と変形処理情報と呼ばれる命令との組み合わせからバージョン2を構成できるようにする一連の命令を作成する手続きを示している。図2の変形処理情報については、すでに説明している。この手続きによって採られる一般的な手法について、以下に説明する。

実行1

まず初めに、バージョン1の処理が行われる。バージョン1は、図2Fの数箇所に表示されている。図5のコードでは、位置0から始まる4文字の文字列が前の位置から始まる4文字の文字列と一致するかどうか尋ねる。位置0の前には何も存在しないことから、当然、答えは「No」である。したがって、位置0は、「実行1」と表示された列に示されるように、カラットを用いてフラグが立てられている。

実行2

実行2では、コードによって、似たような質問、すなわち、位置1から始まる4文字の文字列が前の位置から始まる4文字の文字列と一致するかどうかを尋ねる。答えは「No」であり、位置1にフラグが立てられる。

実行3および実行4

同様に、フラグが位置2および位置3に設定され、図2Fの実行4のようにフラグが立てられる。

【0022】実行5

実行5では、上記とは違った結果が得られる。実行5では、コードにより通常の「位置4から始まる4文字の文字列は、前の位置から始まる4文字の文字列と一致するか？」という問いを出す。実行5として示された列からわかるように、答えは、位置0において、「Yes」で

ある。位置4にはフラグが表示されず、EXTEND関数(図5の16行目)が呼び出されて動作が行われる。EXTEND関数は、一致した文字列の長さを尋ねる。一致したブロックの後ろの次の位置(一致したブロックは位置4~7を占めていることから、次の位置は、位置8である)に4つ前の位置と同じものがあるかどうか尋ねる。答えは、試行TAからわかるように、「Yes」である。次に、一致したブロックの2番目の位置すなわち位置9に、4つ前の位置と同じものがあるかどうか尋ねる。試行TBからわかるように、今度の答えも「Yes」である。この問いは、そのような一致が検出されなくなる試行TEに位置が到達するまで続行される。したがって、EXTEND関数は、位置4~位置11までの8つの位置にわたり一致が見られることを確認した。コードの論理上、拡張された一致の最後の3つの位置、すなわち、「実行5の結果」の中の「BCD」にフラグが立てられる。

【0023】実行6

実行6は、位置12(最も右にあるe)から始まる4文字の並びが前に発生した位置から始まる4文字の文字列と一致しているかどうか尋ねる。ここでは、答えは「No」であり、「実行6」と表示された列に示されるように、位置12にフラグが立てられる。

【0024】結果

図2Fの下部に結果が示されており、「結果」と表示されている。そこで、重要な特徴について2点以下に説明する。第一に、後に説明するように、フラグは、バージョン2の作成に用いられる並びの開始だけでなく、終了も示している。第二に、図示されるように、フラグが立てられない領域がある。バージョン2の作成に用いられる文字列の探索は、こうした領域の外から開始してその領域に侵入することができるが、このような領域内から開始することはない。したがって、探索開始点が削除されることから、全体の探索時間は短縮される。

【0025】バージョン2の処理

実行1~実行4

次に、更新バージョンであるバージョン2の処理が行われる。コードにより、図2Gに示されている位置16、17、18、および19から始まる4文字の並びが以前の位置から始まるものと一致しているかどうか質問される。位置16から始まる4文字の並びが位置0から始まるものと一致しているが、後続の3つが一致していないため、図2Gの上部に示されているように、位置17、18、および19にフラグが立てられる。EXTEND関数では、拡張された一致が全く検出されない。図5のコードは、29行目にジャンプして、図2に出てくる上記の「COPY 4 0」命令を発行する。全体の結果は、「出力」と表示された矢印で示されている。すでに3文字にフラグが立てられており、COPY命令が発行されている。ただし、これらのフラグが立てられた文字

は、バージョン1のフラグが立てられた文字がそうであったように、後に、探索初期設定点として扱われることに注意しなければならない。

【0026】実行5

図5のコードでは、(図2Gの「実行5」と表示されている欄のバージョン2にある)位置20から始まる4文字の並び「x y x y」が前の位置から始まるものと一致しているかどうか尋ねる。コードにより、現在バージョン2に存在するものも含め、各フラグから探索を開始する。さらに、各フラグにおいて、コードによって前後両方向に探索が行われる。(この前後方向の探索については、簡潔化を図るため、これまでの説明では述べられていない。)全体の探索は、図2Gの実行5によって示されている。探索は、(a)各フラグから開始し、(b)4文字を前後両方向にサーチすることから、試行T1～試行T12のすべての4文字の並びが検査される。さらに、バージョン2のフラグを立てられた位置、すなわち、位置17、18、19、および20から、同じような探索が行われる。一致が検出されなかったことから、位置20にあるxにフラグが立てられる。

【0027】実行6

図5のコードでは、(バージョン2にある)位置21から始まる4文字の並び「y x y x」が前の位置から始まるものと一致しているかどうか尋ねる。位置20については、バージョン2に存在するものも含め、各フラグから前後両方向にコード探索を開始する。現在のフラグの状態は、図2Hの右上の部分に示されている。

【0028】実行7

図5のコードでは、(バージョン2にある)位置22から始まる4文字の並び「x y x y」が前の位置から始まるものと一致しているかどうか尋ねる。答えは、位置20において「Yes」である。このため、図2Hに示されるように、位置22にフラグは立てられず、EXTEND関数(図5の16行目)が入力される。このEXTEND関数では、図2Fに関して述べられた方法により、一致が図2Hの位置27まで達していると判断される。したがって、拡張された一致の最後の3つの位置にフラグが立てられ、図の左下の四角枠に示されているような結果となる。このとき、論理上、図5の29行目に進み、2つの命令「ADD 2 x, y」および「COPY 6 20」が発行される。(29行目の引き数「add」および「c」は、ADD命令に関するものであり、引き数「pos」および「len」は、COPY命令に関するものである。)

【0029】実行8

コードでは、位置28から始まる文字の並び「b c d e」が前の位置から始まるものと一致しているかどうか尋ねる。答えは、バージョン1(図2Gの左上を参照)の位置9において「Yes」である。これで、EXTEND関数が呼び出され、一致が追加文字「f」まで延び

ていると判断する。次に、コードにより、29行目において、命令「COPY 5 9」を発行する。これで、図2の4つの命令が作成されたことになる。したがって、バージョン1プラス命令からバージョン2を復元できる。

【0030】特色

前記の概要から、次の原理を読み取ることができる。第一に、ある位置が前に一致した並びを表しているとの判断が下されると、その位置にはフラグが立てられず、その結果、冗長であるとの理由により、該当する位置において後続の探索は開始されない。このため、(可能な場合)探索位置が削減される。第二に、バージョン2は、2種類のデータ源、すなわち、(a)バージョン1またはバージョン2からコピーされた文字列、(b)バージョン2に追加された文字列により構成される。さらに、追加された文字列は、後でコピー動作に使用できる。別の観点から、第一ユーザがバージョン1を保有している場合、また、第二ユーザがバージョン1とバージョン2の両方を保有している場合、第二ユーザが以下の文字列を識別していれば、第一ユーザは、バージョン2の複製を作成できる。

- (a) 第一ユーザのバージョン1からコピーしたもの
- (b) 第一ユーザのバージョン1に付け加えられたもの
- (c) 第一ユーザのバージョン2の複製からコピーされたもの

【0031】第三に、各フラグ表示された位置が、単語表内の1語の開始点を表している。これに類似する語が図1Aに示されている。しかし、図1Aの用語範囲とは異なり、各フラグ表示された位置が表す用語範囲は、極めて多くの並びを表現の対象としている。例えば、バージョン1が以下の通りであるとする。

a b c d e F g h i j k

大文字のF(位置6)にフラグが立てられた場合、次の並びを表している。

f f g f g h f g h i f g h j
f g h i j k

上記の並びは、他のフラグ表示された位置を含むことがある。したがって、単一のフラグ表示された位置は、バージョン2へのコピー対象となる多数の並びを表すこともあり得る。このため、このようなフラグ表示された位置が多数のCOPY命令に現れることもある。例えば、「COPY 3 6」は、「f g h」をコピーすることを意味しており、また、「COPY 5」は、「f g h i j」をコピーすることを意味している。

【0032】機密保護

すでに述べた通り、ADD命令には、バージョン2の内容に関する情報が含まれており、このような情報には、機密保護が必要である。機密保護の1手法が図1Jに示されている。バージョン1内にポインタがくるように、送信された図2のADD命令(「ADD 2 x,

y」) が修正される。修正された命令は、図 1 J に「送信命令」と表示されている。この場合、ポインタが「2」であり、「c」を示している。次に、送信された情報(すなわち、「x」および「y」を表すバイト)が、「c」で始まるデータとの排他的論理和がとられる。図 1 J の左側は、排他的論理和の動作を示している。送信される命令は、「ADD 2 2」プラス排他的論理和の動作結果である。傍受者はバージョン 1 に全くアクセスできないことから、この排他的論理和の結果には、傍受者にとって価値のある情報は一切含まれていない。

【0033】図 1 K の右側にある受信された命令に含まれるデータは、バージョン 1 内の同じデータ、すなわち、「c」で始まるデータとの排他的論理和がとられる。この排他的論理和により、元のデータ、すなわち、「x, y」が回復する。この手続きでは、排他的論理和の動作の特性、すなわち、第一の語と第二の語の排他的論理和をとることにより第三の語を作成する点を利用している。第三の語を第二の語と排他的論理和をとることにより、第一の語を回復できる。このため、以下の送信された命令から、

ADD 2 2 [EX-ORの結果]

目的とする命令である次の命令を得ることができる。

ADD 2 x, y

【0034】重要事項

1. COPY および ADD 命令が現れる順序は、当然、重要である。その順序が全く変われば、異なったバージョン 2 が得られる。別の観点からすれば、文字列自体は一種の情報である。文字列は、命令の組み合わせとして見ることもできる。英語のアルファベットの並べ換えによりワードが生成され情報が伝達されるように、このような並べ換えの仕方に情報が含まれている。文字列の復元を可能にする情報が含まれていれば、当然、順序をバラバラにして送信することもできる。例えば、各命令に番号付けをしてもよい。命令の正しい並びがどのように実行されるかという点とは無関係に、実行により、連結プロセスによってバージョン 2 の複製が作成される。すなわち、図 2 B ~ 図 2 E の例に戻り、

1. 最初の「abcd」が複製(図 2 B)に書き込まれる。「abcd」は、バージョン 1 から得られたものである。

2. 次に、「xy」が連結される(図 2 C)。「xy」は、バージョン 1 から得られたものである。

3. 次に、「xyxyxy」の連結が行われる(図 2 D)。「xyxyxy」は、バージョン 2 から得られたものである。(代わりに、4 組の「xy」をバージョン 1 から取得し、ステップ 2 で連結することもできる。しかし、これではあまり効率的とはいえない。)

4. 「bcdef」の連結が行われる(図 2 E)。「bcdef」は、バージョン 1 から得たものである。

【0035】2. さらに、上記の重要事項 1 に関し、各 COPY および ADD 命令にはコピーまたは追加された部分の長さが含まれていることを発明者から指摘しておく。このため、所定の COPY または ADD 命令について、コピーや追加を行うバージョン 2 内のアドレスを、以前の全体の長さに基づいて直ちに計算することができる。(このような長さに基づきポインタの計算が行われていることから、図 3 のコードにこの状態が示されている。)

【0036】3. 本発明は、任意の種類のデータファイルを更新する際に使用でき、テキストファイル等の特定の種類のファイルに限定されるものではない。本発明は、一般に、バイナリファイルを取り扱うことができる。ファイルには文字が含まれている。各文字は、通常、1 バイトのデータによって表される。1 バイトに 8 ビットが含まれていることから、28、すなわち、256 の想定可能な文字が 1 バイトで表現できる。「テキスト」ファイルでは、このような想定可能な組み合わせをすべて使用するわけではなく、英数字および句読点を表すものだけを使用する。「バイナリ」ファイルでは、想定可能な 256 の組み合わせがすべて使用される。本発明では、バイナリファイルの取り扱いが可能である。これとは対照的に、従来技術のプログラムのデフレンジングでは、テキストファイルしか処理できない。

【0037】4. 本発明は、記憶されているバージョン 1 からバージョン 2 の遠隔地への復元動作に限られるものではない。さらに、プログラムの新規バージョンは、バージョン全体を記憶するのではなく、ADD および COPY 命令を用いて一箇所に記憶することができる。この方法により、記憶スペースが節約される。バージョンの復元を必要とする場合、図 3 のプログラムが実行される。

5. 復元されるファイルのバージョンが、その時点において年代的に早いファイルの後のバージョンである必要はない。例えば、バージョン 1 は、バージョン 2 から復元できる。

【0038】技術上の説明

図 3 および図 5 に示されているコードについて、より技術的な説明を行う。

概要

データファイルは、バイトの並びとみなすことができる。実質的にはすべての現行のコンピュータ上において、1 バイトは、記憶、通信、およびメモリ内のデータの操作によって効率的に圧縮できる最小の自然単位である。「データファイル」という語は、ディスク上に記憶されたファイルを指す場合が多いが、本発明では、そのようなバイトの並びはメインメモリのセグメントにすることも可能である。図 1 では、本発明を用いて 2 台のコンピュータ間でデータを同期化している例を示している。第一に、ソースコンピュータ(翻訳用計算機)上で

は、データの2つのバージョンであるバージョン1およびバージョン2が比較されて、バージョン1をバージョン2に取り込む変形処理情報を作成する。この変形処理情報は、何らかの通信チャネルを介してターゲットコンピュータ（目的計算機）に送信される。次に、ターゲットコンピュータ上では、変形処理情報とバージョン1のローカルコピーを用いてバージョン2を復元する。

【0039】変形処理命令のコーディングとデコーディング

本発明により計算された変形処理情報は、2種類の命令、すなわち、COPYおよびADDの並びにより構成されている。バージョン2の復元中は、COPY命令によりコピー対象となるデータの現存するセグメントの位置と長さが定義され、ADD命令により追加対象となるデータのセグメントが定義される。図2では、バージョン1が「a b c d a b c d a b c d e f g h」のバイトの並びにより構成されている2つのデータファイルの例を示している（ここでは読みやすくするためにスペースを挿入している）。バージョン2は、「a b c d x y x y x y x y b c d e f」の並びにより構成されている。したがって、バージョン1の長さは16、バージョン2の長さは17となっている。図2に示されている変形処理情報では、バージョン1からバージョン2を構成し直すうえで必要となる命令が示されている。ただし、バージョン1の位置は0からコード化され、バージョン2の位置はバージョン1の長さからコード化されるという規約を採用している。例えば、図2の変形処理情報の第三の命令は、20としてコード化されたバージョン2の位置4からの6バイトをコピーするCOPY命令である。

【0040】図3では、一般にバージョンの復元が行われる手続きが示されている。1行目では、変数「n」をバージョン1の長さに初期化する。2行目では、バージョン2の現在位置「c」を0に設定する。3行目では、5行目でエンドオブファイル状態が検出された後に6行目で終了するループを開始する。4行目では、関数readinst()を呼び出して命令を読み取る。7行目では、関数readsize()を用いてコピーするサイズすなわちデータサイズを読み取る。8行目と9行目では、命令がADD命令であるかどうか確認し、そうであれば、現在位置cから開始するバージョン2にデータを読み込む。10行目～15行目では、位置コードで読み取りを行って、バージョン1またはバージョン2から適宜コピー動作を行うことにより、COPY命令の処理を行う。16行目では、バージョン2の現在のコピー位置を新たに復元されたデータの長さ分だけ増加させる。copy()関数は、ディスクメモリ（またはメインメモリ）の1領域から別の領域へデータをコピーする単純関数である。しかし、readinst()、readsize()、readpos()、およびread

ata()関数は、COPYおよびADD命令とそのパラメタがどのようにコード化されるかという点についての具体的な定義に基づいて定義されなければならない。

【0041】図3の手続きを図1の例に当てはめてみると、復号化には4つのステップがあることがわかる。第一ステップでは、位置0から始まるバージョン1から「a b c d」の4バイトをコピーする。第二ステップでは、「x y」の2データバイトを追加する。第三ステップでは、（0から数えて規約により20としてコード化された）位置4から始まる6バイトをバージョン2からコピーする。ただし、このステップの開始時点では、「x y」の2バイトしかコピーに使用できないことから、バージョン2の最初の6バイトである「a b c d x y」が復元されただけである。しかし、データが左から右へコピーされることから、1バイトがコピーされるときは必ず、作成されているはずである。第四および最終ステップでは、バージョン1の位置9から「b c d e f」の5バイトがコピーされる。

【0042】以上で、COPYおよびADD命令のコード化について説明がなされたことになる。このような特種な具体例である上記命令が選択されたのは、発明者の実験により、多くの異なるタイプのデータに対してこのような命令がうまく機能するためである。以上の説明がなされれば、上記のreadinst()、readsize()、およびreadpos()機能は、容易に実行できる。各命令は、制御バイトから開始してコード化が行われる。制御バイトの8ビットは2つの部分に分けられている。最初の4ビットは0～15の数を表しており、各々は、命令の種類と何らかの補助情報に関するコーディングを定義している。以下に、最初の4ビットに関する最初の10の値の一覧を示している。

0: ADD命令

1、2、3: QUICKキャッシュの位置を伴うCOPY命令

4: SELFとしてコード化された位置を伴うCOPY命令

5: HEREとの差としてコード化された位置を伴うCOPY命令

6、7、8、9: RECENTキャッシュからコード化された位置を伴うCOPY命令

QUICKキャッシュは、サイズ768(3×356)の配列である。この配列の各指標には、「p modulo 768」が配列の指標となっているように、新たなCOPY命令の位置の値pが含まれている。このキャッシュは、各COPY命令が（コーディング中に）出力されるか、または、（デコーディング中に）処理された後、更新される。タイプ1、2、または3のCOPY命令は、実際の位置が記憶される配列の指標を計算するために、それぞれ0、256、または512に加算されなければならない0～255の値を有するバイトがその直

後に設定される。

【0043】タイプ4のCOPY命令は、一連のバイトとしてコード化されたコピー位置を有している。タイプ5のCOPY命令は、一連のバイトとしてコード化されたコピー位置と現在位置との差を有している。RECENTキャッシュは、4つの指標を有する配列であり、最新の4つのコピー位置を記憶する。COPY命令が（コーディング中に）出力されるか、または、（デコーディング中に）処理されたときは必ず、そのコピー位置がキャッシュ内の最も古い位置と入れ替わる。タイプ6

（7、8、9）のCOPY命令は、キャッシュの指標1（それぞれ、2、3、4）に対応している。そのコピー位置は、対応するキャッシュ指標に記憶されている位置よりも大きいことが保証されており、その差のみがコード化される。

【0044】タイプ1～9のADD命令およびCOPY命令の場合、制御バイトの2番目の4ビットが、0でなければ、含まれているデータのサイズをコード化する。これらビットが0であれば、各サイズは、次のバイト列としてコード化される。このようなコーディング方法の結果、ADD命令の後に別のADD命令が続くことは決していない。ADD命令のデータサイズが4以下であり、かつ後に続くCOPY命令も小さいことがよくあるが、そのような場合、この2つの命令を単一の制御バイトにマージするうえで、上記の方法は有利である。最初の4ビットである10～15の値は、このような結合された命令の組をコード化する。その場合、制御バイトの2番目の4ビットの最初の2ビットにより、ADD命令のサイズをコード化し、残りの2ビットによりCOPY命令のサイズをコード化する。次に、最初の4ビットの10～15の値の一覧を示している。

10: SELFとしてコード化されたコピー位置を伴うマージされたADD/COPY命令

11: HEREとの差としてコード化されたコピー位置を伴うマージされたADD/COPY命令

12、13、14、15: RECENTキャッシュからコード化されたコピー位置を伴うマージされたADD/COPY命令

【0045】図4は、図2の変形処理情報である4つの命令に関するコード化を示したものである。各命令ごとに、制御バイトの全8ビットが示されている。例えば、2番目の制御バイトの最初の4ビットが0となっているが、これは、バイトによりADD命令がコード化されていることを示している。同じバイトの次の4ビットでは、ADD命令のサイズが2であることを示している。2つのデータバイト「xy」が制御バイトの後に置かれている。3つのCOPY命令は、すべて、SELFタイプを用いてコード化されており、したがって、そのコピー位置は、制御バイトに続く（示されている値を用いて）バイトによりコード化されている。すべての命令の

サイズパラメタは、制御バイト内ですべてコード化できるほど小さい。したがって、この小例は、わずか9バイトを用いるだけで、長さ17バイトのバージョン2が変形処理情報にコード化できることを示している。

【0046】一致セグメントの高速計算

図5は、バージョン2をいくつかのセグメントに分割し、COPY命令およびADD命令としてコード化する方法を示している。ただし、このコーディング方法の場合、長さの短い一致は、少なくともそれが一致するデータとしてコード化を行うスペースを取ることから、有効とはいえない。このため、短い一致を無視するように一致方法を調整することができる。ここで用いる一致の最小の長さは4であり、コーディング手続きの本体にある変数MINによって示されている。

【0047】図5の1行目では、探索テーブルTを初期化して空にする。効率上、Tは、衝突をチェーニングしたハッシュテーブルとして保持されている。このデータ構造は標準型であり、「コンピュータアルゴリズムの設計と分析（The Design and Analysis of Computer Algorithms）」（A. Aho、J. Hopcroft、およびJ. Ullman著、1974年、Addison-Wesley発行）（111～112頁）などのデータ構造およびアルゴリズムの教本に説明されている。

【0048】表Tには、2つのバージョンに一定の選択された位置が記載されている。この位置は、手続きinsert()によって挿入され、最長の一致データセグメントを高速で探索するために、手続きsearch()およびextend()で使用される。2行目および3行目では、手続きprocess()を呼び出し、バージョン1およびバージョン2から位置を選択して表Tに挿入する。バージョン2の処理中には、COPY命令およびADD命令の作成も行われる。

【0049】4～45行目では、手続きprocess()を定義する。5行目と6行目では、変数「n」と「m」をバージョン1および処理中のバージョンの長さに初期化する。7行目では、処理中のバージョンの現在位置「c」を0に初期化する。8行目では、ADD命令のデータの開始を-1（すなわち、なし）に初期化する。9および10行目では、位置cから始まるデータセグメントの最長の一致に関する位置と長さを初期化する。11～42行目では、所定のバージョンを処理するメインループを定義する。12～18行目では、cから始まるデータセグメントと一致する処理済みの最長データセグメントを計算する。12行目で呼び出された手続きsearch()により、長さlen+1の一致を検出する。この手続きでは、位置c+len-1（MIN-1）から始まるMINバイトを表Tの一致している位置を探索するためのキーとして使用する。次に、「seq」のデータと適切なバージョンのデータを逆方向に照

合し、一致が c から $c + len + 1$ までのすべてをカバーしているかどうか確認する。

【0050】このような一致が検出されると、16行目で呼び出された手続き `extend()` により、できるだけ長く前方に一致を延長する。14および15行目は、一致が全くなき、かつ探索ループから外れる場合に相当する。それ以外の場合は、ループの繰返しにより、さらに長い一致を検出する。19および26行目では、現在の一致していない位置 c を表 T に挿入する。`seq` がバージョン1の場合、挿入される実効値は c であり、それ以外の場合は、 $c + \text{バージョン1の長さ}$ (前述したバージョン2のコーディング位置の規約) である。手続き `insert(T, seq, p, 原点)` では、キーとしてバージョン `seq` の位置 p から始まる `MIN` バイトを用いて、コード化された位置「 $p + \text{原点}$ 」を表 T に挿入する。

【0051】20および21行目では、未定義であれば、変数 `add` を c に設定し、そこが一致していないデータのセグメントの開始であることを示す。25行目では、処理ループの次の繰返しを行うために、現在位置を1だけ前方に移動させる。27および28行目は、最長の一致セグメントを検出する事象に相当する。28および29行目では、手続き `writelnst()` を呼び出して、前述した説明に従って、`COPY` 命令および `ADD` 命令を書き出す。`add` が0または正の値であれば、`writelnst()` への最初の2つの引き数により、`ADD` 命令のデータを定義する。第二の2つの引き数では、`COPY` 命令のパラメタを定義する。この手続きの作用は簡単なので、ここでは説明を省略する。30~36行目では、一致したデータセグメントの末尾にある `MIN-1` の位置を表 T に挿入する。37および38行目では、一致した長さの分だけ c を増加し、`add` を-1にリセットする。40および41行目では、処理対象となる十分なデータがない場合に、処理を終了させる。43および44行目では、バージョン2から、一致しない最終データを `ADD` 命令として出力する。

【0052】図6は、表 T に位置を挿入する手続きを示している。図6の2行目は、キーが位置 p から始まる「`seq`」列の `MIN` バイトであることを示している。3行目では、コード化された位置を作成する。4行目では、(`key`, `pos`) の組を表 T に挿入する。図7は、一致を探索する手続きを示している。3および4行目では、現在最長となっている一致の長さの末尾の `MIN-1` バイトから成る探索キーと一致していない新規の1バイトを作成する。5~19行目では、可能であれば、一致長さの延長を試行する。この動作は、作成された探索キーと一致する表 T の全要素を調べ、キー位置に置かれた現在の一致の一部が検討中の要素の対応する部分と一致しているかどうか確認することによって行われる。このテストは、17行目で実行される。この結果が

真であれば、18行目において、`search()` により一致の開始位置が返却される。20行目では、「-1」が返却され、「`len`」よりも長い一致がないことを示す。

【0053】図5の13行目において `search()` への呼び出しが行われた後、一致の長さが、現在、図5の「`len`」の値よりも少なくとも1以上長いことが明らかになっている。図8では、`extend()` 手続きが示されており、この手続きによって、右方向にできるだけ長く一致を延長させる。2~10行目では、一致を探索する正しい列を設定する。11~13行目では、延長を実行する。14行目では、一致した全体の長さを返却する。図2の例に当てはめると、上記の方法により、同じ図に示されている一連の命令が計算できる。図6では、表 T に挿入されるバージョン1およびバージョン2の例の位置が示されている。

【0054】機密保護の強化

2つのバージョン1およびバージョン2と計算による変換処理情報が与えられた場合、変換処理情報の `ADD` 命令のみが、バージョン2から生データを取り入れる。このようなデータは、傍受者がバージョン2に関する貴重な情報を得るのに利用可能である。機密保護の必要性が高いアプリケーションでは、情報の漏洩を防ぐために、`ADD` 命令を次のように修正することができる。まず初めに、各 `ADD` 命令を修正して、その位置から始まるデータセグメントが少なくとも `ADD` データの長さ以上となるようなやり方で、バージョン1から任意に選択される位置となるようなコピーアドレスも持つようにする。そのようにできなければ、`ADD` データはさらに小さい単位に分割することができる。次に、生データは、変換処理情報に出力される前に、このようなデータの選択セグメントからのデータとの排他的論理和がとられる。例えば、図2の同じ `ADD` 命令を用いて、バージョン1の選択位置が2であるとした場合、この位置は、制御バイトの直後に出力され、2データバイト「 xy 」は、出力前に、2バイト「 cd 」との排他的論理和がとられることになる。

【0055】デコーディングの場合、出力前に、各データバイトに対して同じ排他的論理和の演算を行わなければならない。排他的論理和の数学的特性により、あるバイトともうひとつの同じバイトとの排他的論理和が2度とられる場合に、元の値が確実に保持されていることから、このような演算が行われる。しかし、これで、傍受者が、安全であるとみなされたバージョン1のコピーをすでに入手していない限り、変換されたデータバイトから何らかの情報を得ることは確実に不可能になる。図7は、このより安全性の高い方法を用いてコード化が行われた図4の命令を示している。これで、`ADD` 命令は位置2を有していることになる。データバイト「 x 」および「 y 」は、それぞれ、「 c 」および「 d 」と排他的論

理和がとられている。本発明の真の精神および範囲を逸脱しない限り、多くの代替および変更を行うことが可能である。特許証による保護を必要とする発明箇所については、特許請求の範囲に定義されている通りである。

【図面の簡単な説明】

【図 1】本発明がコンピュータ間のデータ通信にどのように用いられるかを示す略図である。

【図 1 A】類推により本発明によって用いられる原理を示す図の A である。

【図 1 B】類推により本発明によって用いられる原理を示す図の B である。

【図 1 C】類推により本発明によって用いられる原理を示す図の C である。

【図 1 D】類推により本発明によって用いられる原理を示す図の D である。

【図 1 E】類推により本発明によって用いられる原理を示す図の E である。

【図 1 F】類推により本発明によって用いられる原理を示す図の F である。

【図 1 G】類推により本発明によって用いられる原理を示す図の G である。

【図 1 H】類推により本発明によって用いられる原理を示す図の H である。

【図 1 I】類推により本発明によって用いられる原理を示す図の I である。

【図 1 J】本発明の機密保護に関する状況を示す図の J である。

【図 2】データセットの 2 つのバージョン例とこのバージョンの片方をもう一方のバージョンに変形するための一連の命令を示す図である。

【図 2 A】図 3 に示されている手続きの動作を示す図の A である。

【図 2 B】図 3 に示されている手続きの動作を示す図の B である。

【図 2 C】図 3 に示されている手続きの動作を示す図の C である。

【図 2 D】図 3 に示されている手続きの動作を示す図の D である。

【図 2 E】図 3 に示されている手続きの動作を示す図の E である。

【図 2 F】図 5 に示されている手続きの動作を示す図の F である。

【図 2 G】図 5 に示されている手続きの動作を示す図の G である。

【図 2 H】図 5 に示されている手続きの動作を示す図の H である。

【図 3】擬似 C 言語による復号化手続きである。

【図 4】図 2 の命令を実際にコード化したものである。

【図 5】コード化の手続きを示す図である。

【図 6】図 5 で使用された INSERT 手続きを示す図である。

【図 7】図 5 で使用された SEARCH 手続きを示す図である。

【図 8】図 5 で使用された EXTEND 手続きを示す図である。

【図 9】図 2 に例示されたバージョン 1 およびバージョン 2 のデータの挿入位置を示す図である。

【図 10】図 4 のコード化された命令の機密保護が考慮されたバージョンである。

付録

Tue Aug 30 09:21:37 1994

|_paragvina.sco.att.com/n/ryphon/g7/kpv/Software/sco/lib/vdelta/vdelta.h|

```

1: #ifndef _VDelta_H
2: #define _VDelta_H 1
3:
4: #ifndef __KPV__
5: #define __KPV__ 1
6:
7: #ifndef __STD_C
8: #ifndef __STDC__
9: #define __STDC__ 1
10: #else
11: #if __cplusplus
12: #define __STDC__ 1
13: #else
14: #define __STDC__ 0
15: #endif /*__cplusplus*/
16: #endif /*__STDC__*/
17: #endif /*__STDC__*/
18:
19: #ifndef _BEGIN_EXTERNS_
20: #if __cplusplus
21: #define _BEGIN_EXTERNS_ extern "C" {
22: #define _END_EXTERNS_ }
23: #else
24: #define _BEGIN_EXTERNS_
25: #define _END_EXTERNS_
26: #endif
27: #endif /*_BEGIN_EXTERNS_*/
28:
29: #ifndef _ARG_
30: #if __STDC__
31: #define _ARG_(X) x
32: #else
33: #define _ARG_(X) ()
34: #endif
35: #endif /*_ARG_*/
36:
37: #ifndef Void_t
38: #if __STDC__
39: #define Void_t void
40: #else
41: #define Void_t char
42: #endif
43: #endif /*Void_t*/
44:
45: #ifndef NIL
46: #define NIL(type) ((type)0)
47: #endif /*NIL*/
48:
49: #endif /*__KPV__*/
50:
51: /* user-supplied functions to do io */
52: typedef struct _vddisc_s Vddisc_t;
53: typedef int (* Vdio_f)_ARG_((int, Void_t*, int, long, Vddisc_t));
54: struct _vddisc_s
55: {
56:     Vdio_f readf; /* to read data */
57:     Vdio_f writef; /* to write data */
58:     long window; /* window size if any */
59: };
60:
61: /* types that can be given to the IO functions */
62: #define VP_SOURCE 1 /* io on the source data */
63: #define VP_TARGET 2 /* io on the target data */
64: #define VP_DELTA 3 /* io on the delta data */
65:
66: /* magic header for delta output */
67: #define VP_MAGIC "vd01"
68:
69: _BEGIN_EXTERNS_
70: extern long vddelta_ARG_((Void_t*, long, Void_t*, long, Vddisc_t));
71: extern long vdupdata_ARG_((Void_t*, long, Void_t*, long, Vddisc_t));
72: _END_EXTERNS_
73: #endif /*_VDelta_H*/

```

Tue Aug 30 09:21:37 1994

|_paragvina.sco.att.com/n/ryphon/g7/kpv/Software/sco/lib/vdelta/vdelta.h|

Wed Aug 10 21:24:44 1994

__error__ rpn_att_ccm1/h/gyphon/sj/kw/software/src/lib/vdelta/vdeltahdr.h 1

```

1: #ifndef _VDELHDR_H
2: #define _VDELHDR_H 1
3:
4: #include "vdelta.h"
5:
6: #if __STD_C
7: #include <stddef.h>
8: #else
9: #include <sys/types.h>
10: #endif
11:
12: #ifdef DEBUG
13: #define _BEGIN_EXTERN_
14: extern int abort();
15: #define _END_EXTERN_
16: #define ASSERT(p) ((p) ? 0 : abort())
17: #define DBTOTAL(t,v) ((t) += (v))
18: #define DBMAX(m,v) ((m) = (m) > (v) ? (m) : (v))
19: #else
20: #define ASSERT(p)
21: #define DBTOTAL(t,v)
22: #define DBMAX(m,v)
23: #endif
24:
25: /* short-hand notations */
26: #define reg register
27: #define uchar unsigned char
28: #define uint unsigned int
29: #define ulong unsigned long
30:
31: /* default window size - Chosen to suit malloc() even on 16-bit machines. */
32: #define MAXWIN ((int)((uint)0 >> 1))
33: #define MAXWINDOW ((int)((uint)0 >> 2))
34: #define DISPLAYWINDOW [MAXWINDOW <= (1<<14) ? (1<<14) : (1<<16)]
35: #define HEADER(w) ((w)/4)
36:
37: #define M_MIN 4 /* min number of bytes to match */
38:
39: /* The hash function is s[0]*alpha^3 + s[1]*alpha^2 + s[2]*alpha + s[3] */
40: #define ALPHA 33
41: #if 0
42: #define A1(x,t) (ALPHA*(x))
43: #define A2(x,t) (ALPHA*ALPHA*(x))
44: #define A3(x,t) (ALPHA*ALPHA*ALPHA*(x))
45: #else /* fast multiplication using shifts and adds */
46: #define A1(x,t) ((t = (x)), (t * (t<<3)))
47: #define A2(x,t) ((t = (x)), (t * (t<<6) + (t<<10)))
48: #define A3(x,t) ((t = (x)), (t * (t<<9) + ((t+(t<<6))<<6) + ((t+(t<<3))<<11)))
49: #endif
50: #define HINIT(h,s,t) ((h = A3(s[0],t)), (h += A2(s[1],t)), (h += A1(s[2],t) + s[3]))
51: #define HNEXT(h,s,t) ((h -= A3(s[-1],t)), (h = A1(h,t) + s[3]))
52:
53: #define EQUAL(s,t) ((s)[0] == (t)[0] && (s)[1] == (t)[1] && \
54: (s)[2] == (t)[2] && (s)[3] == (t)[3])
55:
56: /* Every instruction will start with a control byte.
57: ** For portability, only 8 bits of the byte are used.

```


Wed Aug 10 21:24:44 1994

|_peragrine.xep.att.com:/n/stryphon/q7/Xmv/Software/src/11b/vdelta/vdeltahdr.h |

```

58:  ** The bits are used as follows:
59:  **      1111 ssss.
60:  **      ssss: size of data involved.
61:  **      1111: this defines 15 instruction types:
62:  **          0: an ADD instruction.
63:  **          1,2,3: COPY with K_QUICK addressing scheme.
64:  **          4,5: COPY with K_SELF,K_HERE addressing schemes.
65:  **          6,7,8,9: COPY with K_RECENT addressing scheme.
66:  **          For the above types, ssss is not zero codes the size;
67:  **          otherwise, the size is coded in subsequent bytes.
68:  **          10,11: merged ADD/COPY with K_SELF,K_HERE addressing.
69:  **          12,13,14,15: merged ADD/COPY with K_RECENT addressing.
70:  **          For merged ADD/COPY instructions, ssss is divided into "cc aa"
71:  **          where cc codes the size of COPY and aa codes the size of ADD.
72:  **
73:
74:  #define VD_BITS      8      /* # bits usable in a byte */
75:
76:  #define S_BITS      4      /* bits for the size field */
77:  #define I_BITS      4      /* bits for the instruction type */
78:
79:  /* The below macros compute the coding for a COPY address.
80:  ** There are two caches, a "quick" cache of (K_QTYPE*256) addresses
81:  ** and a revolving cache of K_RTYPE "recent" addresses.
82:  ** First, we look in the quick cache to see if the address is there.
83:  ** If so, we use the cache index as the code.
84:  ** Otherwise, we compute from 0, the current location and
85:  ** the "recent" cache an address that is closest to the being coded address,
86:  ** then code the difference. The type is set accordingly.
87:  **
88:  ** An invariant is 2*K_MERGE + K_QTYPE + 1 == 16
89:  **
90:  #define K_RTYPE      4      /* # of K_RECENT types */
91:  #define K_QTYPE      1      /* # of K_QUICK types */
92:  #define K_MERGE      (K_RTYPE*2) /* # of types allowing add-copy */
93:  #define K_QSIZE      (K_QTYPE<<VD_BITS) /* size of K_QUICK cache */
94:
95:  #define K_QUICK      1      /* start of K_QUICK types */
96:  #define K_SELF      (K_QUICK+K_QTYPE)
97:  #define K_HERE      (K_SELF+1)
98:  #define K_RECENT      (K_HERE+1) /* start of K_RECENT types */
99:
100: #define K_DDECL(quick,recent,rhere) /* cache decls in vdelta */
101: int quick[K_QSIZE]; int recent[K_RTYPE]; int rhere; /*
102: #define K_UDECL(quick,recent,rhere) /* cache decls in vupdate */
103: long quick[K_QSIZE]; long recent[K_RTYPE]; int rhere; /*
104: #define K_INIT(quick,recent,rhere) \
105: { quick[rhere=0] = (1<<7); \
106:   while((rhere += 1) < K_QSIZE) quick[rhere] = rhere + (1<<7); \
107:   recent[rhere=0] = (1<<8); \
108:   while((rhere += 1) < K_RTYPE) recent[rhere] = (rhere+1)*(1<<8); \
109: }
110: #define K_UPDATE(quick,recent,rhere,copy) \
111: { quick[copy%K_QSIZE] = copy; \
112:   if((rhere += 1) >= K_RTYPE) rhere = 0; recent[rhere] = copy; \
113: }
114:
115: #define VD_ISCOPY(k)      ((k) > 0 && (k) < (K_RECENT+K_RTYPE)) ;
116: #define K_ISMERGE(k)      ((k) >= (K_RECENT+K_RTYPE))
117:

```

Wed Aug 10 21:26:44 1994

```

persson_xoo.att.com/h/scrphn/g/krv/software/src/lib/obj-a/rdalad-h
118: #define A_SIZE      ((1<<S_BITS)-1)      /* max local ADD size */
119: #define A_ISLOCAL(s) ((s) <= A_SIZE)      /* can be coded locally */
120: #define A_LPUT(s) (s) /* coded local value */
121: #define A_PUT(s) ((s) - (A_SIZE+1)) /* coded normal value */
122:
123: #define A_ISHERE(i) ((i) < A_SIZE) /* locally coded size */
124: #define A_LGET(i) ((i) - A_SIZE)
125: #define A_GET(s) ((s) + (A_SIZE+1))
126:
127: #define C_SIZE      ((1<<S_BITS)+M_MIN-2) /* max local COPY size */
128: #define C_ISLOCAL(s) ((s) <= C_SIZE) /* can be coded locally */
129: #define C_LPUT(s) ((s) - (M_MIN-1)) /* coded local value */
130: #define C_PUT(s) ((s) - (C_SIZE-1)) /* coded normal value */
131:
132: #define C_ISHERE(i) ((i) < ((1<<S_BITS)-1)) /* size was coded local */
133: #define C_LGET(i) (((i) + ((1<<S_BITS)-1)) - (M_MIN-1))
134: #define C_GET(s) ((s) + (C_SIZE-1))
135:
136: #define K_PUT(k) ((k) <= S_BITS)
137: #define K_GET(i) ((i) >> S_BITS)
138:
139: /* coding merged ADD/COPY instructions */
140: #define A_TINY      2 /* bits for tiny ADD */
141: #define A_TINYSIZE  (1<<A_TINY) /* max tiny ADD size */
142: #define A_ISTINY(s) ((s) <= A_TINYSIZE)
143: #define A_TPUT(s) ((s) - 1)
144: #define A_TGET(i) (((i) & (A_TINYSIZE-1)) + 1)
145:
146: #define C_TINY      2 /* bits for tiny COPY */
147: #define C_TINYSIZE  ((1<<C_TINY) - M_MIN-1) /* max tiny COPY size */
148: #define C_ISTINY(s) ((s) <= C_TINYSIZE)
149: #define C_TPUT(s) (((s) - M_MIN) << A_TINY)
150: #define C_TGET(i) (((i) >> A_TINY) & ((1<<C_TINY)-1)) + M_MIN)
151:
152: #define K_TPUT(k) (((k)+K_MERGE) << S_BITS)
153:
154: #define MEMCPY(to,from,n) \
155:     switch(n) \
156:     { default: memcpy((Void_t*)to,(Void_t*)from,(size_t)n); \
157:       case 7: *to++ = *from++; \
158:       case 6: *to++ = *from++; \
159:       case 5: *to++ = *from++; \
160:       case 4: *to++ = *from++; \
161:       case 3: *to++ = *from++; \
162:       case 2: *to++ = *from++; \
163:       case 1: *to++ = *from++; \
164:       case 0: break; \
165:     }
166:
167: /* Below here is code for a buffered I/O subsystem to speed up I/O */
168: #define I_SHIFT      7
169: #define I_MORE      (1<<I_SHIFT) /* continuation bit */
170: #define I_CODE(n) ((uchar)((n)&(I_MORE-1))) /* get lower bits */
171:
172: /* structure to do buffered IO */
173: typedef struct _vdio_s
174: { uchar* next;
175:   uchar* endb;
176:   vddisc_t* disc;
177:

```

Wed Aug 10 21:24:44 1994

paragrace.xop.att.com/n/gryphon/g7/kpv/Software/src/lib/vdelta-vdeltahdr.h

```

178: long      here;
179: uchar     buf[1024];
180: } Vdio_t;
181:
182: #define READP(io) ((io)->disc->readf)
183: #define WRITEP(io) ((io)->disc->writef)
184: #define BUF(io) ((io)->buf)
185: #define BUFSIZE(io) sizeof((io)->buf)
186: #define NEXT(io) ((io)->next)
187: #define ENDB(io) ((io)->endb)
188: #define DISC(io) ((io)->disc)
189: #define HERE(io) ((io)->here)
190: #define RINIT(io, disc) ((io)->endb = ((io)->next = (io)->buf), \
191:                          (io)->here = 0, (io)->disc = (disc) )
192: #define WINIT(io, disc) ((io)->endb = ((io)->next = (io)->buf) + BUFSIZE(io), \
193:                          (io)->here = 0, (io)->disc = (disc) )
194: #define REMAIN(io) (ENDP(io) - NEXT(io))
195: #define VDPUTC(io, c) ((REMAIN(io) > 0 || (*_vdfilbuf)(io) > 0) ? \
196:                       (int)((io)->next-- = (c)) : -1)
197: #define VDGETC(io) ((REMAIN(io) > 0 || (*_vdfilbuf)(io) > 0) ? \
198:                    (int)((io)->next++) : -1)
199:
200: typedef struct _vdbufio_s
201: { int(* vdfilbuf)_ARG_((Vdio_t*));
202:   int(* vdfilbuf)_ARG_((Vdio_t*));
203:   ulong(* vdfgetu)_ARG_((Vdio_t*, ulong));
204:   int(* vdfputu)_ARG_((Vdio_t*, ulong));
205:   int(* vdfread)_ARG_((Vdio_t*, uchar*, int));
206:   int(* vdfwrite)_ARG_((Vdio_t*, uchar*, int));
207: } Vdbufio_t;
208: #define _Vdfilbuf _Vdbufio.vdfilbuf
209: #define _Vdfilbuf _Vdbufio.vdfilbuf
210: #define _Vdfgetu _Vdbufio.vdfgetu
211: #define _Vdfputu _Vdbufio.vdfputu
212: #define _Vdfread _Vdbufio.vdfread
213: #define _Vdfwrite _Vdbufio.vdfwrite
214:
215: _BEGIN_EXTERNS_
216: extern Vdbufio_t _Vdbufio;
217: extern Void_t* memcpy_ARG_((Void_t*, const Void_t*, size_t));
218: extern Void_t* malloc_ARG_((size_t));
219: extern void free_ARG_((Void_t*));
220: _END_EXTERNS_
221:
222: #endif /* _VDLHDR_H */

```

Wed Sep 28 08:36:11 1994

paragrace.xop.att.com/n/gryphon/g7/kpv/Software/src/lib/vdelta-vdelta.c

```

1: #include "vdelhdr.h"
2:
3: /* Compute a transformation that takes source data to target data
4:  *
5:  * Written by (Kiem-Phong Vo, kpv@research.att.com, 5/20/94
6:  */
7:
8: #ifdef DEBUG
9: long  S_copy, S_add; /* amount of input covered by COPY and ADD */
10: long  M_copy, M_add; /* # of COPY and ADD instructions */
11: long  M_copy, M_add; /* max size of a COPY or ADD instruction */
12: long  M_merge; /* # of merged instructions */
13: #endif
14:
15: #define MERGEABLE(a, c, k) ((a) > 0 && A_ISINST(a) && \
16:                             (c) > 0 && C_ISINST(c) && \
17:                             (k) >= R_SELF)
18:
19: typedef struct _match_s Match_t;
20: typedef struct _table_s Table_t;
21: struct _match_s
22: { Match_t* next; /* linked list ptr */
23: };
24: struct _table_s
25: { Vdio_t io; /* io structure */
26:   uchar* src; /* source string */
27:   int n_src;
28:   uchar* tar; /* target string */
29:   int n_tar;
30:   K_BDECL(quick, recent, there); /* address caches */
31:   Match_t* base; /* base of elements */
32:   int size; /* size of hash table */
33:   Match_t* table; /* hash table */
34: };
35:
36: /* encode and output delta instructions */
37: #if _STD_C
38: static vdfputinst(Table_t* tab, uchar* begs, uchar* here, Match_t* match, int n_copy)
39: #else
40: static vdfputinst(tab, begs, here, match, n_copy)
41: #endif
42: { Table_t* tab;
43:   uchar* begs; /* ADD data if any */
44:   uchar* here; /* current location */
45:   Match_t* match; /* best match if any */
46:   int n_copy; /* length of match */
47:   #endif
48:   {
49:     reg int n_add, i_add, i_copy, k_type;
50:     reg int n, c_addr, copy, best, d;
51:
52:     n_add = begs ? here - begs : 0; /* add size */
53:     c_addr = (here - tab->tar) + tab->n_src; /* current address */
54:     k_type = 0;
55:
56:     if (match) /* process the COPY instruction */
57:     {
58:       /**/DBTOTAL(M_copy, 1); DBTOTAL(S_copy, n_copy); DBMAX(M_copy, n_copy);
59:       best = copy = match - tab->base;
60:       k_type = R_SELF;

```

Wed Sep 28 08:36:11 1994

paragrine.zoo.att.com/n/stryphen/q7/kpv/softwara/src/lib/vdio_t.c/vddelra.c

```

60:     if((d = c_addr - copy) < best)
61:     {
62:         best = d;
63:         k_type = K_HERE;
64:     }
65:     for(n = 0; n < K_RTYPE; ++n)
66:     {
67:         if((d = copy - tab->recent[n]) < 0 || d >= best)
68:             continue;
69:         best = d;
70:         k_type = K_RECENT+n;
71:     }
72:     if(best >= I_MORE && tab->quick[n = copy%K_SIZE] == copy)
73:     {
74:         for(d = K_QTYPE-1; d > 0; --d)
75:             if(n >= (d<<VD_BITS))
76:                 break;
77:         best = n - (d<<VD_BITS); /*/ASSERT(best < (1<<VD_BITS));
78:         k_type = K_QUICK*d;
79:     }
80:     /*/ASSERT(best >= 0);
81:     /*/ASSERT((k_type^K_HERE) < (1<<I_BITS));
82:     /* update address caches */
83:     K_UPDATE(tab->quick, tab->recent, tab->where, copy);
84:     /* see if mergable to last ADD instruction */
85:     if(MERGABLE(n_add, n_copy, k_type))
86:     {
87:         /*/DBTOTAL(M_merge, 1);
88:         i_add = K_TPUT(k_type); A_TPUT(n_add); C_TPUT(n_copy);
89:     }
90:     else
91:     {
92:         i_copy = K_PUT(k_type);
93:         if(C_ISLOCAL(n_copy))
94:             i_copy |= C_LPUT(n_copy);
95:     }
96:     if(n_add > 0)
97:     {
98:         /*/DBTOTAL(M_add, 1); DBTOTAL(S_add, n_add); DBMAX(M_add, n_add);
99:         if(!MERGABLE(n_add, n_copy, k_type))
100:             i_add = A_ISLOCAL(n_add) ? A_LPUT(n_add) : 0;
101:         if(VDPUTC((Vdio_t*)tab, i_add) < 0)
102:             return -1;
103:         if(!A_ISLOCAL(n_add) &&
104:            (*_Vdputu)((Vdio_t*)tab, (ulong)A_PUT(n_add)) < 0)
105:             return -1;
106:         if((*_Vdwrite)((Vdio_t*)tab, begs, n_add) < 0)
107:             return -1;
108:     }
109:     if(n_copy > 0)
110:     {
111:         if(!MERGABLE(n_add, n_copy, k_type) && VDPUTC((Vdio_t*)tab, i_copy) < 0)
112:             return -1;
113:         if(!C_ISLOCAL(n_copy) &&
114:            (*_Vdputu)((Vdio_t*)tab, (ulong)C_PUT(n_copy)) < 0)
115:             return -1;
116:     }
117:
118:

```

Wed Sep 28 08:36:11 1994

perexline.zoo.att.com/p/ucyphon/g7/kpy/Software/src/lib/vdelta/vddrite.c

```

119:         if(k_type >= K_QUICK && k_type < (K_QUICK+E_QTYPE) )
120:         {
121:             if(VDPUTC((Vdio_t*)tab,(uchar)best) < 0 )
122:                 return -1;
123:         }
124:         else
125:         {
126:             if((*_Vdputu)((Vdio_t*)tab, (ulong)best) < 0 )
127:                 return -1;
128:         }
129:     }
130:     else
131:     {
132:         if((*_Vdfilebuf)((Vdio_t*)tab) < 0)
133:             return -1;
134:     }
135:     return 0;
136: }
137:
138: /* Fold a string */
139: #if __STD_C
140: static vdfold(Table_t* tab, int output)
141: #else
142: static vdfold(tab, output)
143: Table_t* tab;
144: int output;
145: #endif
146: {
147:     reg_ulong      key, n;
148:     reg_uchar      *s, *am, *ends, *ss, *heads;
149:     reg_Match_t    *m, *list, *curr, *bestm;
150:     reg_uchar      *add, *endfold;
151:     reg_int         head, len, n_src = tab->n_src;
152:     reg_int         size = tab->size;
153:     reg_uchar      *src = tab->src, *tar = tab->tar;
154:     reg_Match_t     *base = tab->base, **table = tab->table;
155:
156:     if(!output)
157:     {
158:         if(tab->n_src < M_MIN)
159:             return 0;
160:         endfold = (s = src) + tab->n_src;
161:         curr = base;
162:     }
163:     else
164:     {
165:         endfold = (s = tar) + tab->n_tar;
166:         curr = base+n_src;
167:         if(tab->n_tar < M_MIN)
168:             return vdfutinst(tab,s,endfold,NIL(Match_t*),0);
169:     }
170:
171:     add = NIL(uchar*);
172:     bestm = NIL(Match_t*);
173:     len = M_MIN-1;
174:     MINIT(key,s,n);
175:     for(;;)
176:     {
177:         for(;;) /* search for the longest match */
178:         {
179:             if(!m = table[key/size])
180:                 goto endsearch;
181:             list = m = m->next; /* head of list */
182:             if(bestm) /* skip over past elements */

```

Wed Sep 28 08:16:11 1994

csregina.zoo.att.com/n/gr/phon/g/kpv/Software/src/lib/dfa/rdelta.c

```

179:         for(;;)
180:         {
181:             if(m >= bestm*len)
182:                 break;
183:             if((m = m->next) == list)
184:                 goto endsearch;
185:         }
186:
187:     head = len - (M_MIN-1); /* header before the match */
188:     head = s+head;
189:     for(;;)
190:     {
191:         if((n = m-base) < n_src)
192:         {
193:             if(n < head)
194:                 goto next;
195:             sm = src - n;
196:         }
197:         else
198:         {
199:             if((n = n_src) < head)
200:                 goto next;
201:             sm = src + n;
202:         }
203:
204:         /* make sure that the M_MIN bytes match */
205:         if(!EQUAL(head,sm))
206:             goto next;
207:
208:         /* make sure this is a real match */
209:         for(sm = head, ss = s; ss < head; )
210:             if(*sm++ != *ss++)
211:                 goto next;
212:
213:         ss += M_MIN;
214:         sm += M_MIN;
215:         enda = endfold;
216:         if((n-base) < n_src && (n = (src+n_src)-sm) < (enda-s))
217:             enda = s+n;
218:         for(ss = enda; ss < enda; ++ss, --sm)
219:             if(*sm != *ss)
220:                 goto extend;
221:         goto extend;
222:
223:     next: if((m = m->next) == list)
224:             goto endsearch;
225:     }
226:
227: extend: bestm = m-head;
228:     n = len;
229:     len = ss-s;
230:     if(ss >= endfold) /* already match everything */
231:         goto endsearch;
232:
233:     /* check for a longer match */
234:     ss = M_MIN-1;
235:     if(len == n+1)
236:         HNEXT(key,ss,n);
237:     else
238:         HINIT(key,ss,n);
239: }
240:
241: endsearch:
242:     if(bestm)

```

Wed Sep 28 08:36:11 1994

```

238:         if(output && vdupinst(tab,add,s,bestm,len) < 0)
239:             return -1;
240:
241:         /* add a sufficient number of suffices */
242:         enda = (s + len);
243:         ss = enda - (M_MIN-1);
244:         if(!output)
245:             curm = base + (ss-src);
246:         else
247:             curm = base + n_src + (ss-tar);
248:
249:         len = M_MIN-1;
250:         add = NIL(uchar*);
251:         bestm = NIL(Match_t*);
252:     }
253:     else
254:     {
255:         if(!add)
256:             add = s;
257:         ss = s;
258:         enda = (s + 1); /* add one prefix */
259:
260:         if(enda > (endfold - (M_MIN-1)))
261:             enda = endfold - (M_MIN-1);
262:
263:         if(ss < enda) for(;;) /* add prefixes/suffices */
264:         {
265:             n = keysize;
266:             if(!m = table[n])
267:                 curm->next = curm;
268:             else
269:             {
270:                 curm->next = m->next;
271:                 n->next = curm;
272:             }
273:             table[n] = curm++;
274:             if((ss + 1) >= enda)
275:                 break;
276:             MNEXT(key,ss,n);
277:         }
278:
279:         if(s > endfold-M_MIN) /* too short to match */
280:         {
281:             if(!add)
282:                 add = s;
283:             break;
284:         }
285:         MNEXT(key,s,n);
286:     }
287:
288:     if(output) /* flush output */
289:         return vdupinst(tab,add,endfold,NIL(Match_t*),0);
290:     return 0;
291: }
292:
293: #if __STD_C
294: long vddelta(Void_t* src, long n_src, Void_t* tar, long n_tar, vddisc_t* disc)
295: #else
296: long vddelta(src, n_src, tar, n_tar, disc)
297: Void_t* src; /* source string if not NULL */
298: long n_src; /* length of source data */

```

Wed Sep 28 08:36:11 1994

| ~~parag~~ ~~ne_xxx_att_nom~~ / ~~n/oryphon/g7/kpv/Software/src/lib/vdc/ta/vddelta.c~~.

```

298: void _=      tar;      /* target string if not NULL */
299: long         n_tar;     /* length of target data */
300: Vddisc_t* disc; /* IO discipline */
301: #endif
302: {
303:     reg int      size, k, n, window;
304:     reg long     p;
305:     Table_t      tab;
306:
307:     if(!disc || !disc->write) {
308:         if(!disc->read) && ((n_tar > 0 && !tar) || (n_src > 0 && !src)) {
309:             return -1;
310:         }
311:         if(n_tar < 0)
312:             return -1;
313:         if(n_src < 0)
314:             n_src = 0;
315:
316:         tab.n_src = tab.n_tar = tab.size = 0;
317:         tab.tar = tab.src = NIL(uchar*);
318:         tab.base = NIL(Match_t*);
319:         tab.table = NIL(Match_t*);
320:         WINIT(&tab.io.disc);
321:
322:         if(disc->window <= 0)
323:             window = DEFAULTWINDOW;
324:         else if(disc->window > MAXWINDOW)
325:             window = MAXWINDOW;
326:         else window = (int)disc->window;
327:         if((long>window > n_tar)
328:            window = (int)n_tar;
329:         if(n_src > 0 && (long>window > n_src)
330:            window = (int)n_src;
331:
332:         /* try to allocate working space */
333:         while(window > 0)
334:         {
335:             /* space for the target string */
336:             size = (n_tar == 0 || tar) ? 0 : window;
337:             if((long)size > n_tar)
338:                 size = (int)n_tar;
339:             if(size > 0 && !(tab.tar = (uchar*)malloc(size*sizeof(uchar))))
340:                 goto reduce_window;
341:
342:             /* space for sliding header or source string */
343:             if(n_src <= 0) /* compression only */
344:                 size = tar ? 0 : HEADER(window);
345:             else /* differencing */
346:             {
347:                 size = src ? 0 : window;
348:                 if((long)size > n_src)
349:                     size = (int)n_src;
350:             }
351:             if(size > 0 && !(tab.src = (uchar*)malloc(size*sizeof(uchar))))
352:                 goto reduce_window;
353:
354:             /* space for the hash table elements */
355:             size = window < n_tar ? window : (int)n_tar;
356:             if(n_src <= 0)
357:                 size += window < n_tar ? HEADER(window) : 0;
358:             else size += window < n_src ? window : (int)n_src;
359:             if(!(tab.base = (Match_t*)malloc(size*sizeof(Match_t))))

```


Wed Sep 28 08:16:11 1994

paragrine.zoo.att.com/n/gryphon/q7/kmy/Software/src/lib/rdelta/rdelta.c;

```

358:         goto reduce_window;
359:
360:     /* space for the hash table */
361:     n = size/2;
362:     do (size = n); while((n <= n-1) != 0);
363:     if(size < 64)
364:         size = 64;
365:     while(! (tab.table = (Match_t*) malloc(size*sizeof(Match_t))) )
366:         if((size >= 1) <= 0)
367:             goto reduce_window;
368:
369:     /* if get here, successful */
370:     tab.size = size-1;
371:     break;
372:
373: reduce_window:
374:     if(tab.tar)
375:     {
376:         free((Void_t*) tab.tar);
377:         tab.tar = NIL(char*);
378:     }
379:     if(tab.src)
380:     {
381:         free((Void_t*) tab.src);
382:         tab.src = NIL(char*);
383:     }
384:     if(tab.base)
385:     {
386:         free((Void_t*) tab.base);
387:         tab.base = NIL(Match_t*);
388:     }
389:     if((window >= 1) <= 0)
390:         return -1;
391:
392: /* amount processed */
393: n = 0;
394:
395: /* output magic bytes and sizes */
396: for(k = 0; VD_MAGIC[k]; k++)
397: {
398:     if(! _Vdwrite)(tab.io, (uchar*) VD_MAGIC[k] != k ||
399:         !_Vdputu)(tab.io, (ulong)n_tar) <= 0 ||
400:         !_Vdputu)(tab.io, (ulong)n_src) <= 0 ||
401:         !_Vdputu)(tab.io, (ulong>window) <= 0 )
402:         goto done;
403:
404: /* do one window at a time */
405: while(n < n_tar)
406: {
407:     /* prepare the source string */
408:     if(n_src <= 0) /* data compression */
409:     {
410:         if(n <= 0)
411:             tab.n_src = 0;
412:         else
413:             size = HEADER(window);
414:         if(tab.tar)
415:             tab.src = tab.tar + tab.n_tar - size;
416:         else
417:             memcpy((Void_t*) tab.src,
418:                 (Void_t*) (tab.tar + tab.n_tar - size),
419:                 size );
420:         tab.n_src = size;
421:     }
422: }

```

Wed Sep 28 08:36:11 1994

paragins.zop.att.noni/n/gryphon/g7/kpv/software/src/lib/vda-2a/vddc1a.c

```

418:     else /* data differencing */
419:     {
420:         if(n < n_src)
421:         {
422:             if(n+window > n_src)
423:                 p = n_src-window;
424:             else
425:                 p = n;
426:             if(src)
427:                 tab.src = (uchar*)src + p;
428:             else
429:             {
430:                 size = (*disc->readf)(VD_SOURCE, tab.src,
431:                                         window, p, disc);
432:                 if(size != window)
433:                     goto done;
434:             }
435:         } /* else use last window */
436:         tab.n_src = window;
437:     }
438:     /* prepare the target string */
439:     size = (n_tar-n) < window ? (int)(n_tar-n) : window;
440:     tab.n_tar = size;
441:     if(tar)
442:         tab.tar = (uchar*)tar + n;
443:     else
444:     {
445:         size = (*disc->readf)(VD_TARGET, tab.tar, size, (long)n, disc);
446:         if((long)size != tab.n_tar)
447:             goto done;
448:     }
449:     /* reinitialise table before processing */
450:     for(k = tab.size; k >= 0; --k)
451:         tab.table[k] = NII(Match_t);
452:     K_INIT(tab.quick, tab.recent, tab.xhere);
453:     if(tab.n_src > 0 && vdfold(stab, 0) < 0)
454:         goto done;
455:     if(vdfold(stab, 1) < 0)
456:         goto done;
457:     n += size;
458: }
459:
460: done:
461: if(!tar && tab.tar)
462:     free((Void_t*)tab.tar);
463: if(tab.src && (n_src <= 0 && !tar) || (n_src > 0 && !src))
464:     free((Void_t*)tab.src);
465: if(tab.base)
466:     free((Void_t*)tab.base);
467: if(tab.table)
468:     free((Void_t*)tab.table);
469:
470: return n;
471: }

```

Sun Aug 14 11:53:45 1994

/peregine.ica.att.com/p/gyphon/s/kpv/software/src/lib/vdel-s/vdupdate.c

```

1: #include "vdelhdr.h"
2:
3:
4: /*      Apply the transformation source->target to reconstruct target
5:  **      This code is designed to work even if the local machine has
6:  **      word size smaller than that of the machine where the delta
7:  **      was computed. A requirement is that "long" on the local
8:  **      machine must be large enough to hold source and target sizes.
9:  **      It is also assumed that if an array is given, the size of
10:  **      that array in bytes must be storable in an "int". This is
11:  **      used in various cast from "long" to "int".
12:  **
13:  **      Written by (Kiem-Phong Vo, kp@research.att.com, 5/20/94
14:  */
15: typedef struct _table_s
16: {
17:     Vdio_t      io;          /* io structure          */
18:     uchar*      src;         /* source string        */
19:     uchar*      tar;         /* target string        */
20:     long        n_tar;
21:     long        s_org;       /* start of window in source */
22:     long        t_org;       /* start of window in target */
23:     uchar       data[128];   /* buffer for data transferring */
24:     char        s_alloca;    /* 1 if source was allocated */
25:     char        t_alloca;    /* 1 if target was allocated */
26:     char        compress;    /* 1 if compressing only */
27:     K_UDECL(quick.recent.rhere); /* address caches */
28: } Table_t;
29:
30: #if __STD_C
31: static vdunfold(Table_t* tab)
32: #else
33: static vdunfold(tab)
34:     Table_t* tab;
35: #endif
36: {
37:     reg long      size, copy;
38:     reg int       inst, k_type, n, r;
39:     reg uchar     *tar, *src, *to, *fr;
40:     reg long      t, c_addr, n_tar, n_src;
41:     reg Vdio_f     readf, writef;
42:     reg Vddisc_t*  disc;
43:
44:     n_tar = tab->n_tar;
45:     tar = tab->tar;
46:     n_src = tab->n_src;
47:     src = tab->src;
48:
49:     disc = tab->io.disc;
50:     readf = disc->readf;
51:     writef = disc->writef;
52:
53:     for(t = 0, c_addr = n_src; t < n_tar; )
54:     {
55:         if((inst = VDGETC((Vdio_t*)tab)) < 0)
56:             return -1;
57:         k_type = K_GET(inst);
58:         if(!VD_ISCOPY(k_type))
59:             if(K_ISMERGE(k_type)) /* merge/add instruction */

```

Sun Aug 14 11:55:45 1994

_peract:na.200.att.com/n/xyvshnp/s7/Kpy/Software/xxn/lib/vdelt/vdupdate.c:

```

60:         size = A_GETC(inst);
61:     else if(A_ISHERE(inst)) /* locally coded ADD size */
62:         size = A_LGET(inst);
63:     else /* non-local ADD size */
64:     {
65:         if((size = VDGETC((Vdio_t*)tab)) < 0)
66:             return -1;
67:         if(size >= I_MORE 46
68:            {size = (long)(*_Vdgetu)((Vdio_t*)tab, size)) < 0)
69:             return -1;
70:         size = A_GET(size);
71:     }
72:     if((t+size) > n_tar) /* out of sync */
73:         return -1;
74:     c_addr += size;
75:     /* copy data from the delta stream to target */
76:     for(;;)
77:     {
78:         if(!tar)
79:         {
80:             if((long)(n - sizeof(tab->data)) > size)
81:                 n = (int)size;
82:             if(*_Vdread)((Vdio_t*)tab, tab->data, n) != n
83:                 return -1;
84:             r = (*_writef)(VD_TARGET,
85:                          (Vdio_t*)tab->data, n,
86:                          tab->t_org+t, disc);
87:             if(r != n)
88:                 return -1;
89:         }
90:         else
91:         {
92:             n = (int)size;
93:             if(*_Vdread)((Vdio_t*)tab, tar+t, n) != n)
94:                 return -1;
95:         }
96:         t += n;
97:         if((size -= n) <= 0)
98:             break;
99:     }
100:     if(K_ISMERGE(k_type))
101:     {
102:         size = C_GETC(inst);
103:         k_type = K_MERGE;
104:         goto do_copy;
105:     }
106:     else
107:     {
108:         if(C_ISHERE(inst)) /* locally coded COPY size */
109:             size = C_LGET(inst);
110:         else
111:         {
112:             if((size = VDGETC((Vdio_t*)tab)) < 0)
113:                 return -1;
114:             if(size >= I_MORE 46
115:                {size = (long)(*_Vdgetu)((Vdio_t*)tab, size)) < 0)
116:                 return -1;
117:             size = C_GET(size);
118:         }
119:     }
120: do_copy:
121:     if((t+size) > n_tar) /* out of sync */
122:         return -1;

```


Sun Aug 14 11:55:45 1994

```

#pragma GCC optimize ("O3")
#pragma GCC optimize ("O2")
#pragma GCC optimize ("O1")
#pragma GCC optimize ("O0")

```

```

176:                                     copy, disc);
177:                                     if(x != n)
178:                                         return -1;
179:                                     x = (*writef)(VD_TARGET,
180:                                     (Void_t*)tab->data, n
181:                                     );
182:                                     tab->t_org=t, disc);
183:                                     }
184:                                     if(x != n)
185:                                         return -1;
186:                                     t += n;
187:                                     if((size -= n) <= 0)
188:                                         break;
189:                                     copy += n;
190:                                     }
191:                                     }
192:                                     else /* copy from target data */
193:                                     {
194:                                         copy -= n_src;
195:                                         if(copy >= t || (copy+size) > n_tar) /* out-of-sync */
196:                                             return -1;
197:                                         for(;;) /* allow for copying overlapped data */
198:                                         {
199:                                             reg long a, s;
200:                                             if((s = t-copy) > size)
201:                                                 s = size;
202:                                             if(tar)
203:                                             {
204:                                                 to = tar+t; fr = tar+copy; n = (int)s;
205:                                                 MEMCPY(to, fr, n);
206:                                                 t += n;
207:                                                 goto next;
208:                                             }
209:                                             /* hard read/write */
210:                                             a = copy;
211:                                             for(;;)
212:                                             {
213:                                                 if((long)(n = sizeof(tab->data)) > s)
214:                                                     n = (int)s;
215:                                                 x = (*readf)(VD_TARGET,
216:                                                 (Void_t*)tab->data, n,
217:                                                 a + tab->t_org, disc);
218:                                                 if(x != n)
219:                                                     return -1;
220:                                                 x = (*writef)(VD_TARGET,
221:                                                 (Void_t*)tab->data, n,
222:                                                 t + tab->t_org, disc);
223:                                                 if(x != n)
224:                                                     return -1;
225:                                                 t += n;
226:                                                 if((s -= n) <= 0)
227:                                                     break;
228:                                                 a += n;
229:                                             }
230:                                         }
231:                                         next: if((size -= s) == 0)
232:                                             break;
233:                                     }
234:                                     }
235:                                     }
236:                                     }
237:                                     }
238:                                     }
239:                                     }
240:                                     }
241:                                     }
242:                                     }
243:                                     }
244:                                     }
245:                                     }
246:                                     }
247:                                     }
248:                                     }
249:                                     }
250:                                     }
251:                                     }
252:                                     }
253:                                     }
254:                                     }
255:                                     }
256:                                     }
257:                                     }
258:                                     }
259:                                     }
260:                                     }
261:                                     }
262:                                     }
263:                                     }
264:                                     }
265:                                     }
266:                                     }
267:                                     }
268:                                     }
269:                                     }
270:                                     }
271:                                     }
272:                                     }
273:                                     }
274:                                     }
275:                                     }
276:                                     }
277:                                     }
278:                                     }
279:                                     }
280:                                     }
281:                                     }
282:                                     }
283:                                     }
284:                                     }
285:                                     }
286:                                     }
287:                                     }
288:                                     }
289:                                     }
290:                                     }
291:                                     }
292:                                     }
293:                                     }
294:                                     }
295:                                     }
296:                                     }
297:                                     }
298:                                     }
299:                                     }
300:                                     }
301:                                     }
302:                                     }
303:                                     }
304:                                     }
305:                                     }
306:                                     }
307:                                     }
308:                                     }
309:                                     }
310:                                     }
311:                                     }
312:                                     }
313:                                     }
314:                                     }
315:                                     }
316:                                     }
317:                                     }
318:                                     }
319:                                     }
320:                                     }
321:                                     }
322:                                     }
323:                                     }
324:                                     }
325:                                     }
326:                                     }
327:                                     }
328:                                     }
329:                                     }
330:                                     }
331:                                     }
332:                                     }
333:                                     }
334:                                     }
335:                                     }
336:                                     }
337:                                     }
338:                                     }
339:                                     }
340:                                     }
341:                                     }
342:                                     }
343:                                     }
344:                                     }
345:                                     }
346:                                     }
347:                                     }
348:                                     }
349:                                     }
350:                                     }
351:                                     }
352:                                     }
353:                                     }
354:                                     }
355:                                     }
356:                                     }
357:                                     }
358:                                     }
359:                                     }
360:                                     }
361:                                     }
362:                                     }
363:                                     }
364:                                     }
365:                                     }
366:                                     }
367:                                     }
368:                                     }
369:                                     }
370:                                     }
371:                                     }
372:                                     }
373:                                     }
374:                                     }
375:                                     }
376:                                     }
377:                                     }
378:                                     }
379:                                     }
380:                                     }
381:                                     }
382:                                     }
383:                                     }
384:                                     }
385:                                     }
386:                                     }
387:                                     }
388:                                     }
389:                                     }
390:                                     }
391:                                     }
392:                                     }
393:                                     }
394:                                     }
395:                                     }
396:                                     }
397:                                     }
398:                                     }
399:                                     }
400:                                     }
401:                                     }
402:                                     }
403:                                     }
404:                                     }
405:                                     }
406:                                     }
407:                                     }
408:                                     }
409:                                     }
410:                                     }
411:                                     }
412:                                     }
413:                                     }
414:                                     }
415:                                     }
416:                                     }
417:                                     }
418:                                     }
419:                                     }
420:                                     }
421:                                     }
422:                                     }
423:                                     }
424:                                     }
425:                                     }
426:                                     }
427:                                     }
428:                                     }
429:                                     }
430:                                     }
431:                                     }
432:                                     }
433:                                     }
434:                                     }
435:                                     }
436:                                     }
437:                                     }
438:                                     }
439:                                     }
440:                                     }
441:                                     }
442:                                     }
443:                                     }
444:                                     }
445:                                     }
446:                                     }
447:                                     }
448:                                     }
449:                                     }
450:                                     }
451:                                     }
452:                                     }
453:                                     }
454:                                     }
455:                                     }
456:                                     }
457:                                     }
458:                                     }
459:                                     }
460:                                     }
461:                                     }
462:                                     }
463:                                     }
464:                                     }
465:                                     }
466:                                     }
467:                                     }
468:                                     }
469:                                     }
470:                                     }
471:                                     }
472:                                     }
473:                                     }
474:                                     }
475:                                     }
476:                                     }
477:                                     }
478:                                     }
479:                                     }
480:                                     }
481:                                     }
482:                                     }
483:                                     }
484:                                     }
485:                                     }
486:                                     }
487:                                     }
488:                                     }
489:                                     }
490:                                     }
491:                                     }
492:                                     }
493:                                     }
494:                                     }
495:                                     }
496:                                     }
497:                                     }
498:                                     }
499:                                     }
500:                                     }
501:                                     }
502:                                     }
503:                                     }
504:                                     }
505:                                     }
506:                                     }
507:                                     }
508:                                     }
509:                                     }
510:                                     }
511:                                     }
512:                                     }
513:                                     }
514:                                     }
515:                                     }
516:                                     }
517:                                     }
518:                                     }
519:                                     }
520:                                     }
521:                                     }
522:                                     }
523:                                     }
524:                                     }
525:                                     }
526:                                     }
527:                                     }
528:                                     }
529:                                     }
530:                                     }
531:                                     }
532:                                     }
533:                                     }
534:                                     }
535:                                     }
536:                                     }
537:                                     }
538:                                     }
539:                                     }
540:                                     }
541:                                     }
542:                                     }
543:                                     }
544:                                     }
545:                                     }
546:                                     }
547:                                     }
548:                                     }
549:                                     }
550:                                     }
551:                                     }
552:                                     }
553:                                     }
554:                                     }
555:                                     }
556:                                     }
557:                                     }
558:                                     }
559:                                     }
560:                                     }
561:                                     }
562:                                     }
563:                                     }
564:                                     }
565:                                     }
566:                                     }
567:                                     }
568:                                     }
569:                                     }
570:                                     }
571:                                     }
572:                                     }
573:                                     }
574:                                     }
575:                                     }
576:                                     }
577:                                     }
578:                                     }
579:                                     }
580:                                     }
581:                                     }
582:                                     }
583:                                     }
584:                                     }
585:                                     }
586:                                     }
587:                                     }
588:                                     }
589:                                     }
590:                                     }
591:                                     }
592:                                     }
593:                                     }
594:                                     }
595:                                     }
596:                                     }
597:                                     }
598:                                     }
599:                                     }
600:                                     }
601:                                     }
602:                                     }
603:                                     }
604:                                     }
605:                                     }
606:                                     }
607:                                     }
608:                                     }
609:                                     }
610:                                     }
611:                                     }
612:                                     }
613:                                     }
614:                                     }
615:                                     }
616:                                     }
617:                                     }
618:                                     }
619:                                     }
620:                                     }
621:                                     }
622:                                     }
623:                                     }
624:                                     }
625:                                     }
626:                                     }
627:                                     }
628:                                     }
629:                                     }
630:                                     }
631:                                     }
632:                                     }
633:                                     }
634:                                     }
635:                                     }
636:                                     }
637:                                     }
638:                                     }
639:                                     }
640:                                     }
641:                                     }
642:                                     }
643:                                     }
644:                                     }
645:                                     }
646:                                     }
647:                                     }
648:                                     }
649:                                     }
650:                                     }
651:                                     }
652:                                     }
653:                                     }
654:                                     }
655:                                     }
656:                                     }
657:                                     }
658:                                     }
659:                                     }
660:                                     }
661:                                     }
662:                                     }
663:                                     }
664:                                     }
665:                                     }
666:                                     }
667:                                     }
668:                                     }
669:                                     }
670:                                     }
671:                                     }
672:                                     }
673:                                     }
674:                                     }
675:                                     }
676:                                     }
677:                                     }
678:                                     }
679:                                     }
680:                                     }
681:                                     }
682:                                     }
683:                                     }
684:                                     }
685:                                     }
686:                                     }
687:                                     }
688:                                     }
689:                                     }
690:                                     }
691:                                     }
692:                                     }
693:                                     }
694:                                     }
695:                                     }
696:                                     }
697:                                     }
698:                                     }
699:                                     }
700:                                     }
701:                                     }
702:                                     }
703:                                     }
704:                                     }
705:                                     }
706:                                     }
707:                                     }
708:                                     }
709:                                     }
710:                                     }
711:                                     }
712:                                     }
713:                                     }
714:                                     }
715:                                     }
716:                                     }
717:                                     }
718:                                     }
719:                                     }
720:                                     }
721:                                     }
722:                                     }
723:                                     }
724:                                     }
725:                                     }
726:                                     }
727:                                     }
728:                                     }
729:                                     }
730:                                     }
731:                                     }
732:                                     }
733:                                     }
734:                                     }
735:                                     }
736:                                     }
737:                                     }
738:                                     }
739:                                     }
740:                                     }
741:                                     }
742:                                     }
743:                                     }
744:                                     }
745:                                     }
746:                                     }
747:                                     }
748:                                     }
749:                                     }
750:                                     }
751:                                     }
752:                                     }
753:                                     }
754:                                     }
755:                                     }
756:                                     }
757:                                     }
758:                                     }
759:                                     }
760:                                     }
761:                                     }
762:                                     }
763:                                     }
764:                                     }
765:                                     }
766:                                     }
767:                                     }
768:                                     }
769:                                     }
770:                                     }
771:                                     }
772:                                     }
773:                                     }
774:                                     }
775:                                     }
776:                                     }
777:                                     }
778:                                     }
779:                                     }
780:                                     }
781:                                     }
782:                                     }
783:                                     }
784:                                     }
785:                                     }
786:                                     }
787:                                     }
788:                                     }
789:                                     }
790:                                     }
791:                                     }
792:                                     }
793:                                     }
794:                                     }
795:                                     }
796:                                     }
797:                                     }
798:                                     }
799:                                     }
800:                                     }
801:                                     }
802:                                     }
803:                                     }
804:                                     }
805:                                     }
806:                                     }
807:                                     }
808:                                     }
809:                                     }
810:                                     }
811:                                     }
812:                                     }
813:                                     }
814:                                     }
815:                                     }
816:                                     }
817:                                     }
818:                                     }
819:                                     }
820:                                     }
821:                                     }
822:                                     }
823:                                     }
824:                                     }
825:                                     }
826:                                     }
827:                                     }
828:                                     }
829:                                     }
830:                                     }
831:                                     }
832:                                     }
833:                                     }
834:                                     }
835:                                     }
836:                                     }
837:                                     }
838:                                     }
839:                                     }
840:                                     }
841:                                     }
842:                                     }
843:                                     }
844:                                     }
845:                                     }
846:                                     }
847:                                     }
848:                                     }
849:                                     }
850:                                     }
851:                                     }
852:                                     }
853:                                     }
854:                                     }
855:                                     }
856:                                     }
857:                                     }
858:                                     }
859:                                     }
860:                                     }
861:                                     }
862:                                     }
863:                                     }
864:                                     }
865:                                     }
866:                                     }
867:                                     }
868:                                     }
869:                                     }
870:                                     }
871:                                     }
872:                                     }
873:                                     }
874:                                     }
875:                                     }
876:                                     }
877:                                     }
878:                                     }
879:                                     }
880:                                     }
881:                                     }
882:                                     }
883:                                     }
884:                                     }
885:                                     }
886:                                     }
887:                                     }
888:                                     }
889:                                     }
890:                                     }
891:                                     }
892:                                     }
893:                                     }
894:                                     }
895:                                     }
896:                                     }
897:                                     }
898:                                     }
899:                                     }
900:                                     }
901:                                     }
902:                                     }
903:                                     }
904:                                     }
905:                                     }
906:                                     }
907:                                     }
908:                                     }
909:                                     }
910:                                     }
911:                                     }
912:                                     }
913:                                     }
914:                                     }
915:                                     }
916:                                     }
917:                                     }
918:                                     }
919:                                     }
920:                                     }
921:                                     }
922:                                     }
923:                                     }
924:                                     }
925:                                     }
926:                                     }
927:                                     }
928:                                     }
929:                                     }
930:                                     }
931:                                     }
932:                                     }
933:                                     }
934:                                     }
935:                                     }
936:                                     }
937:                                     }
938:                                     }
939:                                     }
940:                                     }
941:                                     }
942:                                     }
943:                                     }
944:                                     }
945:                                     }
946:                                     }
947:                                     }
948:                                     }
949:                                     }
950:                                     }
951:                                     }
952:                                     }
953:                                     }
954:                                     }
955:                                     }
956:                                     }
957:                                     }
958:                                     }
959:                                     }
960:                                     }
961:                                     }
962:                                     }
963:                                     }
964:                                     }
965:                                     }
966:                                     }
967:                                     }
968:                                     }
969:                                     }
970:                                     }
971:                                     }
972:                                     }
973:                                     }
974:                                     }
975:                                     }
976:                                     }
977:                                     }
978:                                     }
979:                                     }
980:                                     }
981:                                     }
982:                                     }
983:                                     }
984:                                     }
985:                                     }
986:                                     }
987:                                     }
988:                                     }
989:                                     }
990:                                     }
991:                                     }
992:                                     }
993:                                     }
994:                                     }
995:                                     }
996:                                     }
997:                                     }
998:                                     }
999:                                     }
1000:                                    }

```

Sun Aug 14 11:55:45 1994

| peragrine.zoo.att.com/p/gryphon/s7/kpv/Software/src/lib/vdclra/vdupdate.c |

```

233: }
234:
235: return 0;
236: }
237:
238: #if __SIC_C
239: long vdupdate(Void_t* src, long n_src, Void_t* tar, long n_tar, vddisc_t* disc)
240: #else
241: long vdupdate(src, n_src, tar, n_tar, disc)
242: Void_t*      src; /* source string if any */
243: long         n_src; /* length of src */
244: Void_t*      tar; /* target space if any */
245: long         n_tar; /* size of tar */
246: vddisc_t* disc;
247: #endif
248: {
249:     Table_t tab;
250:     uchar *data, magic[8];
251:     int n, r;
252:     long t, p, window;
253:     Vdio_f readf, writef;
254:
255:     if(!disc || !disc->readf || !disc->writef)
256:         return -1;
257:     readf = disc->readf;
258:     writef = disc->writef;
259:
260:     /* initialize I/O buffer */
261:     RINIT(&tab.io, disc);
262:
263:     /* check magic header */
264:     data = (uchar*)(VD_MAGIC);
265:     for(n = 0; data[n]; ++n)
266:         ;
267:     if(!readf(&tab.io, magic, n) != n)
268:         return -1;
269:     for(n = 1; n >= 0; --n)
270:         if(data[n] != magic[n])
271:             return -1;
272:
273:     /* get true target size */
274:     if((t = (long)(*_Vdgetu)(&tab.io, 0)) < 0 || (tar && n_tar != t))
275:         return -1;
276:     n_tar = t;
277:
278:     /* get true source size */
279:     if((t = (long)(*_Vdgetu)(&tab.io, 0)) < 0 || (src && n_src != t))
280:         return -1;
281:     n_src = t;
282:
283:     /* get window size */
284:     if((window = (long)(*_Vdgetu)(&tab.io, 0)) < 0)
285:         return -1;
286:
287:     tab.compress = n_src == 0 ? 1 : 0;
288:
289:     /* if we have space, it'll be faster to unfold */
290:     tab.tar = tab.sro = NIL(uchar);
291:     tab.t_alloc = tab.e_alloc = 0;
292:

```

Sun Aug 14 11:53:45 1994

paragrine.xco.att.com/n/grvphon/q7/kpv/Software/src/lib/vdcl-a/vdupdate.c

```

293: if(n_tar > 0 && !tar && window < (long)MAXINT)
294:     n = (int>window;
295: else     n = 0;
296: if(n > 0 && (tab.tar = (uchar*)malloc(n*sizeof(uchar))) )
297:     tab.t_alloc = 1;
298:
299: if(n_src > 0 && !src && window < (long)MAXINT)
300:     n = (int>window;
301: else if(n_src == 0 && window < n_tar && !tar && HEADER(window) < (long)MAXINT
302: |
303:     n = (int)HEADER(window);
304: else     n = 0;
305: if(n > 0 && (tab.src = (uchar*)malloc(n*sizeof(uchar))) )
306:     tab.s_alloc = 1;
307:
308: for(t = 0; t < n_tar; )
309: {
310:     tab.t_org = t; /* current location in target stream */
311:     if(n_src == 0) /* data compression */
312:     {
313:         tab.s_org = 0;
314:         if(t == 0)
315:             tab.n_src = 0;
316:         else
317:         {
318:             tab.n_src = HEADER(window);
319:             p = t - tab.n_src;
320:             if(tar)
321:                 tab.src = (uchar*)tar + p;
322:             else if(tab.src)
323:             {
324:                 n = (int)tab.n_src;
325:                 if(tab.tar)
326:                 {
327:                     data = tab.tar + tab.n_tar - n;
328:                     memcpy((Void_t*)tab.src, (Void_t*)data,
329: /n);
330:                 }
331:                 else
332:                 {
333:                     r = (*readf)(VD_TARGET, tab.src, n, p, di
334: if(r != n)
335:                     goto done;
336:                 }
337:             }
338:         }
339:     }
340:     else /* data differencing */
341:     {
342:         tab.n_src = window;
343:         if(t < n_src)
344:         {
345:             if((t+window) > n_src)
346:                 p = n_src-window;
347:             else
348:                 p = t;
349:             tab.s_org = p;
350:             if(src)
351:                 tab.src = (uchar*)src + p;
352:             else if(tab.src)
353:             {
354:                 n = (int)tab.n_src;
355:                 r = (*readf)(VD_SOURCE, tab.src, n, p, disc);
356:                 if(r != n)
357:                     goto done;
358:             }
359:         }
360:     }
361:     tab.t = t;
362:     t += tab.n_tar;
363: }
364:
365: done:
366: if(tab.t_alloc)
367:     free((Void_t*)tab.tar);
368: if(tab.s_alloc)
369:     free((Void_t*)tab.src);
370:
371: return t;
372:

```

Sun Aug 14 11:53:45 1994

paragrine.xco.att.com/n/grvphon/q7/kpv/Software/src/lib/vdcl-a/vdupdate.c

```

350:     }
351: }
352:
353: if(tar)
354:     tab.tar = (uchar*)tar+t;
355: tab.n_tar = window < (n_tar-t) ? window : (n_tar-t);
356:
357: X_INIT(tab.quick, tab.recent, tab.there);
358: if(vdunfod(stab) < 0)
359:     goto done;
360:
361: if(!tar && tab.tar)
362: {
363:     p = (*writef)(VD_TARGET, (Void_t*)tab.tar, (int)tab.n_tar, t, dis
364: 0);
365:     if(p != tab.n_tar)
366:         goto done;
367: }
368:
369: t += tab.n_tar;
370: }
371:
372: done:
373: if(tab.t_alloc)
374:     free((Void_t*)tab.tar);
375: if(tab.s_alloc)
376:     free((Void_t*)tab.src);
377:
378: return t;
379:

```


Sun Aug 14 12:54:34 1994

/usr/src/king_200_att.com/n/cryphon/g2/kpv/software/src/lib/vdelta/vdio.c;

```

1: #include "vdelhdr.h"
2:
3:
4: #if __STD_C
5: static _vdfilbuf(reg Vdio_t* io)
6: #else
7: static _vdfilbuf(io)
8: reg Vdio_t* io;
9: #endif
10: { reg int n;
11:
12:   if((n = (*READF(io))(VD_DELTA,BUF(io),BUFSIZE(io),HERE(io),DISC(io))) > 0)
13:   {
14:     ENDB(io) = (NEXT(io) = BUF(io)) + n;
15:     HERE(io) += n;
16:     return n;
17:   }
18:
19: #if __STD_C
20: static _vdflebuf(reg Vdio_t* io)
21: #else
22: static _vdflebuf(io)
23: reg Vdio_t* io;
24: #endif
25: { reg int n;
26:
27:   if((n = NEXT(io) - BUF(io)) > 0 &&
28:      (*WRITEF(io))(VD_DELTA,BUF(io),n,HERE(io),DISC(io)) != n)
29:     return -1;
30:
31:   HERE(io) += n;
32:   NEXT(io) = BUF(io);
33:   return BUFSIZE(io);
34: }
35:
36: #if __STD_C
37: static ulong _vdgetu(reg Vdio_t* io, reg ulong v)
38: #else
39: static ulong _vdgetu(io,v)
40: reg Vdio_t* io;
41: reg ulong v;
42: #endif
43: { reg int c;
44:
45:   for(v &= I_MORE-1;;)
46:   {
47:     if((c = VDCYC(io)) < 0)
48:       return (ulong)(-1);
49:     if(!c&I_MORE)
50:       return ((v<<I_SHIFT) | c);
51:     v = (v<<I_SHIFT) | (c & (I_MORE-1));
52:   }
53:
54: #if __STD_C
55: static _vdputu(reg Vdio_t* io, ulong v)
56: #else
57: static _vdputu(io, v)
58: reg Vdio_t* io;
59: reg ulong v;

```

Sun Aug 14 11:54:34 1994

paragrine.xoo.att.com/n/gryphon/g7/xpv/Software/src/lib/vdelta/vdio.c

```

60: #endif
61: {
62:     reg uchar    *s, *next;
63:     reg int      len;
64:     uchar        c(sizeof(ulong)+1);
65:
66:     s = next = io[sizeof(c)-1];
67:     *s = I_CODE(v);
68:     while((v >= I_SHIFT) )
69:         *--s = I_CODE(v)(I_MORE;
70:     len = (next-s) - 1;
71:
72:     if(REMAIN(io) < len && _vdfilbuf(io) <= 0)
73:         return -1;
74:
75:     next = io->next;
76:     switch(len)
77:     {
78:     default: memcpy((Void_t*)next, (Void_t*)s, len); next += len; break;
79:     case 3: *next++ = *s++;
80:     case 2: *next++ = *s++;
81:     case 1: *next++ = *s;
82:     }
83:     io->next = next;
84:
85:     return len;
86: }
87:
88: #if __STD_C
89: static _vdfread(Vdio_t* io, reg uchar* s, reg int n)
90: #else
91: static _vdfread(io, s, n)
92:     Vdio_t* io;
93:     reg uchar* s;
94:     reg int n;
95: #endif
96: {
97:     reg uchar*    next;
98:     reg int       r, m;
99:
100:     for(m = n; m > 0; )
101:     {
102:         if((r = REMAIN(io)) <= 0 && (r = _vdfilbuf(io)) <= 0)
103:             break;
104:         if(r > m)
105:             r = m;
106:
107:         next = io->next;
108:         MEMCPY(s, next, r);
109:         io->next = next;
110:
111:         m -= r;
112:     }
113:     return n-m;
114: }
115:
116: #if __STD_C
117: static _vdfwrite(Vdio_t* io, reg uchar* s, reg int n)
118: #else
119: static _vdfwrite(io, s, n)
120:     Vdio_t* io;

```

Sun Aug 14 11:54:34 1994

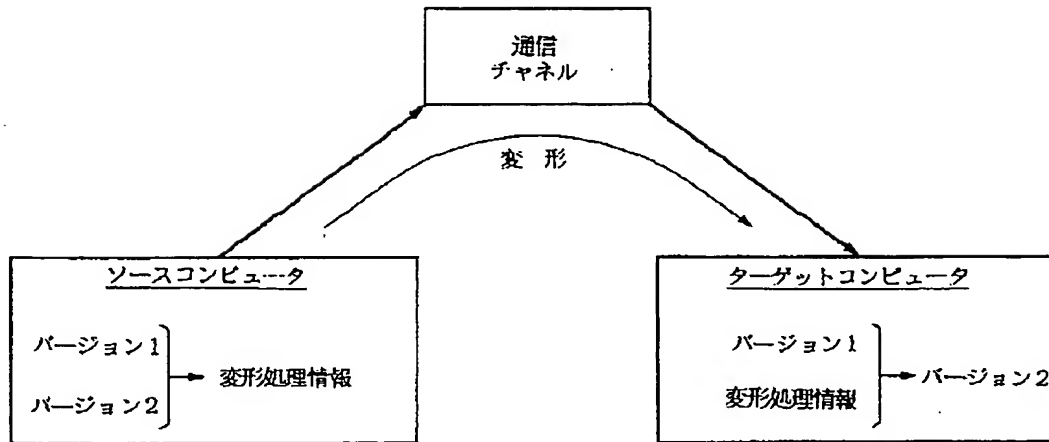
paragrine.xoo.att.com/n/gryphon/g7/xpv/Software/src/lib/vdelta/vdio.c

```

120:     reg uchar*    s;
121:     reg int       n;
122: #endif
123: {
124:     reg uchar*    next;
125:     reg int       w, m;
126:
127:     for(m = n; m > 0; )
128:     {
129:         if((w = REMAIN(io)) <= 0 && (w = _vdfilbuf(io)) <= 0)
130:             break;
131:         if(w > m)
132:             w = m;
133:
134:         next = io->next;
135:         MEMCPY(next, s, w);
136:         io->next = next;
137:
138:         m -= w;
139:     }
140:     return n-m;
141: }
142:
143: vdbufio_t vdbufio =
144: {
145:     _vdfilbuf,
146:     _vdfilbuf,
147:     _vdgetu,
148:     _vdputu,
149:     _vdfread,
150:     _vdfwrite

```

【図 1】



【図 4】

```

01000100 0
00000010 x y
01000110 20
01000101 9

```

【図 1 A】

船 → When buying coconuts, make sure that they are crack free and
 位置 → 1 2 3 4 5 6 7 8 9 10 11
 have no mold on them. Shake them to make sure that they are
 12 13 14 15 16 17 18 19 20 21 22 23 24
 heavy with water. Now hold a coconut in one hand over a sink
 25 26 27 28 29 30 31 32 33 34 35 36 37
 and hit it around the center with the claw end of a hammer.
 38 39 40 41 42 43 44 45 46 47 48 49 50

【図 1 B】

Hawaiian all
 When buying coconuts, make sure that they are crack free and
 1 2 3 4 5 6 7 8 9 10 11
 have no mold on them. Shake them to make sure that they are
 12 13 14 15 16 17 18 19 20 21 22 23 24
 heavy with water. Now hold a coconut in one hand over a sink
 25 26 27 28 29 30 31 32 33 34 35 36 37
 a brick
 and hit it around the center with the claw end of a hammer.
 38 39 40 41 42 43 44 45 46 47 48 49 50

単語表

1 When	16 them	31 it
2 buying	17 Shake	32 around
3 coconuts	18 to	33 the
4 make	19 heavy	34 center
5 sure	20 with	35 claw
6 that	21 water	36 end
7 they	22 Now	37 of
8 are	23 hold	38 hammer
9 crack	24 a	
10 free	25 in	
11 and	26 one	
12 have	27 hand	
13 no	28 over	
14 mold	29 sink	
15 on	30 hit	

【図 1 C】

When buying Hawaiian coconuts, make sure all are crack free and
 1 2 3 4 5 6 7 8 9 10 11
 have no mold on them. Shake them to make sure that they are
 12 13 14 15 16 17 18 19 20 21 22 23 24
 heavy with water. Now hold a coconut in one hand over a sink
 25 26 27 28 29 30 31 32 33 34 35 36 37
 and hit it around the center with a brick.
 38 39 40 41 42 43 44 45 46

【図 1 D】

When buying coconuts, make sure that they are crack free and
 1 2 3 4 5 6 7 8 9 10 11
 have no mold on them. Shake them to make sure that they are
 12 13 14 15 16 17 18 19 20 21 22 23 24
 heavy with water. Now hold a coconut in one hand over a sink
 25 26 27 28 29 30 31 32 33 34 35 36 37
 and hit it around the center with the claw end of a hammer.
 38 39 40 41 42 43 44 45 46 47 48 49 50

単語表

1 When	16 them	31 it
2 buying	17 Shake	32 around
3 coconuts	18 to	33 the
4 make	19 heavy	34 center
5 sure	20 with	35 claw
6 that	21 water	36 end
7 they	22 Now	37 of
8 are	23 hold	38 hammer
9 crack	24 a	39 Hawaiian
10 free	25 in	40 all
11 and	26 one	41 brick
12 have	27 hand	
13 no	28 over	
14 mold	29 sink	
15 on	30 hit	

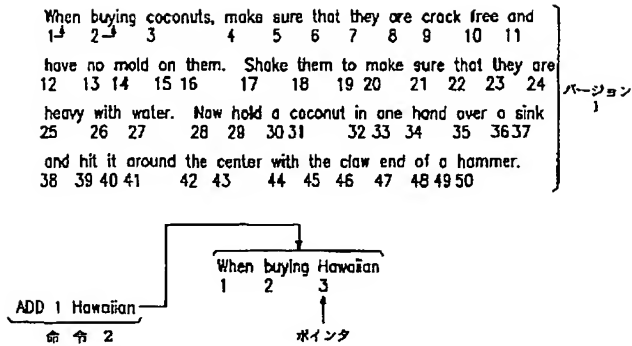
COPY 2 1

命令 1

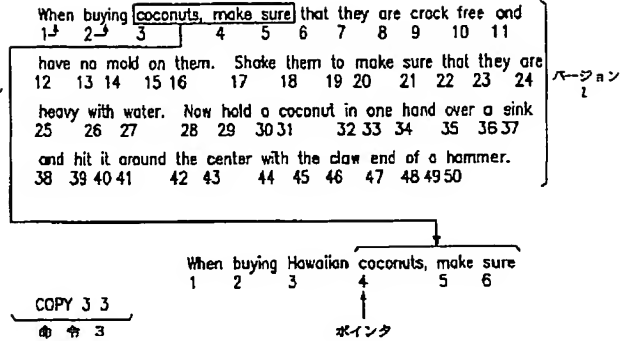
When buying
1
2

ポインタ

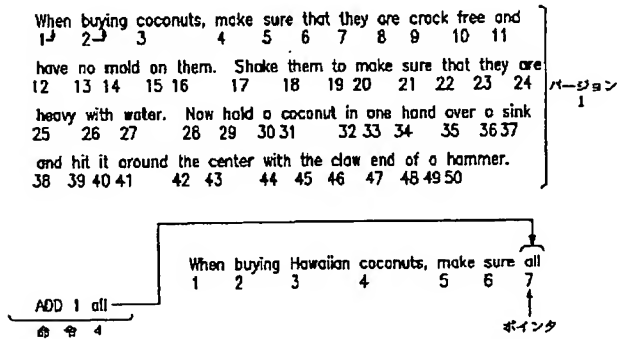
【図 1 E】



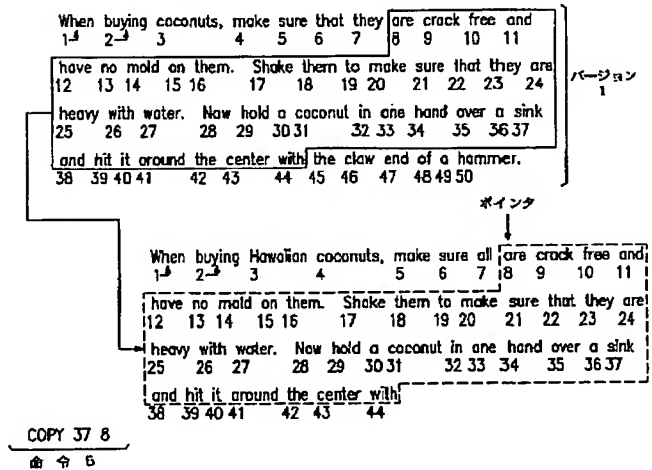
【図 1 F】



【図 1 G】



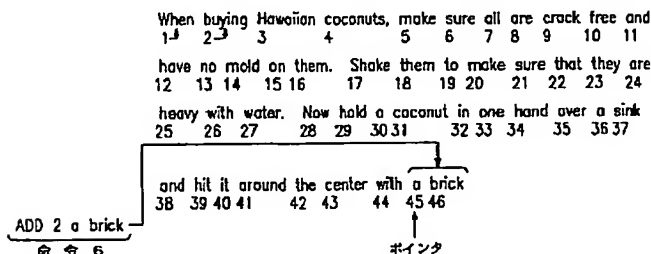
【図 1 H】



【図 1 I】

When buying coconuts, make sure that they are crack free and
 1 2 3 4 5 6 7 8 9 10 11
 have no mold on them. Shake them to make sure that they are
 12 13 14 15 16 17 18 19 20 21 22 23 24
 heavy with water. Now hold a coconut in one hand over a sink
 25 26 27 28 29 30 31 32 33 34 35 36 37
 and hit it around the center with the claw end of a hammer.
 38 39 40 41 42 43 44 45 46 47 48 49 50

バージョン 1



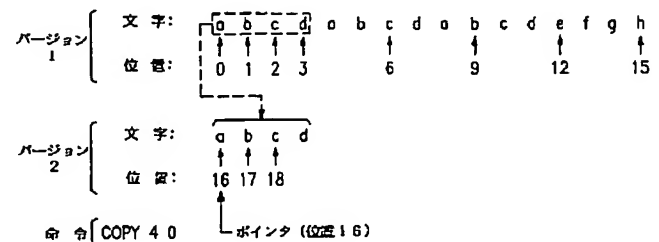
【図 2】

バージョン 1 { a b c d a b c d a b c d e f g h
 バージョン 2 { a b c d x y x y x y x y b c d e f

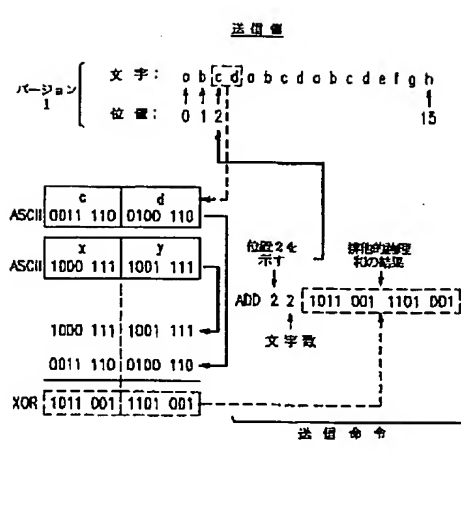
変形処理情報

1. COPY	4	0
2. ADD	2	x, y
3. COPY	6	20
4. COPY	5	9

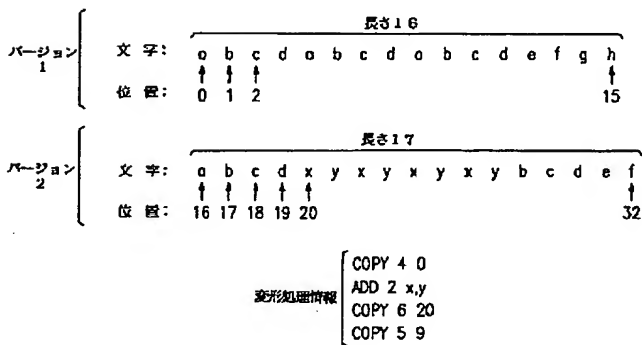
【図 2 B】



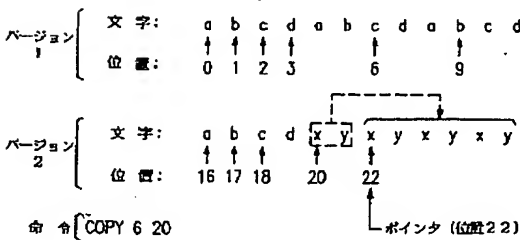
【図1J】



【図2A】

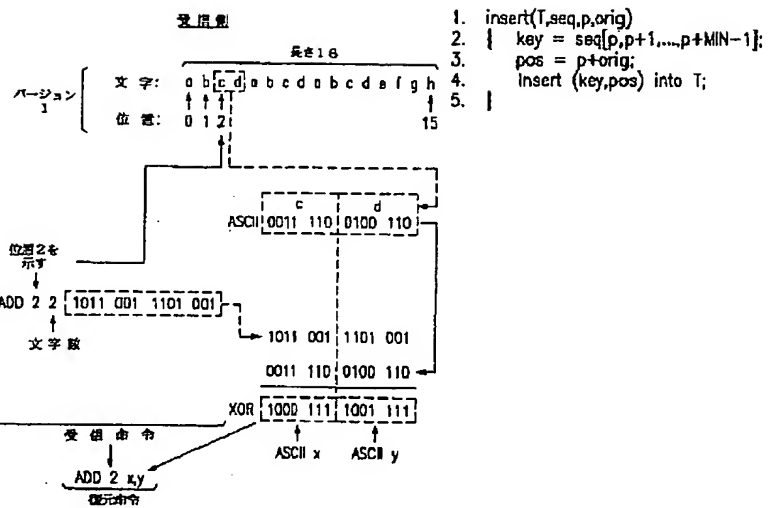


【図2D】

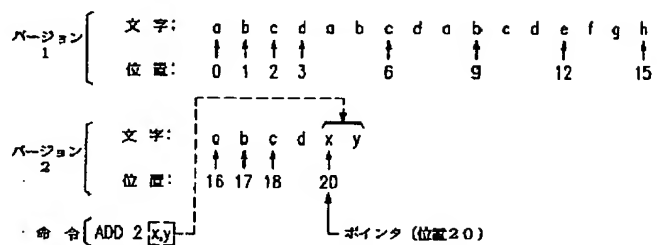


【図9】

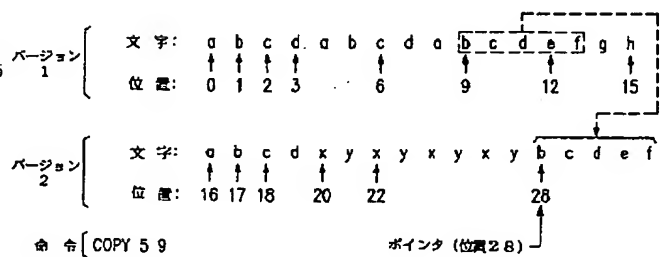
【図6】



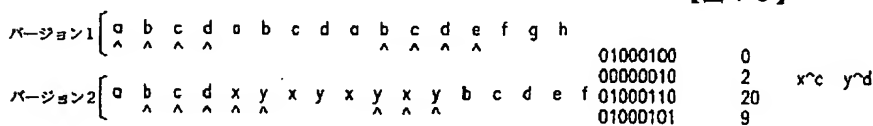
【図2C】



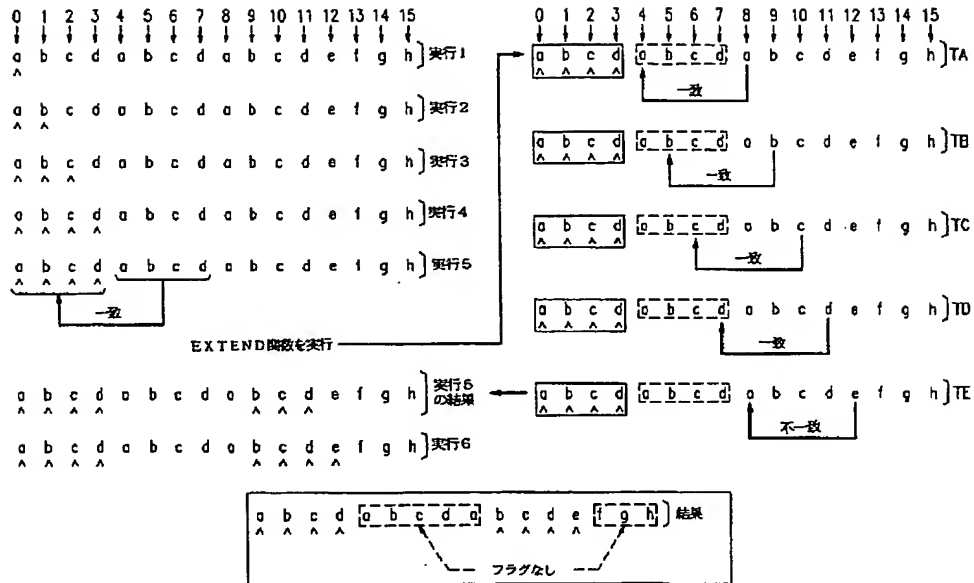
【図2E】



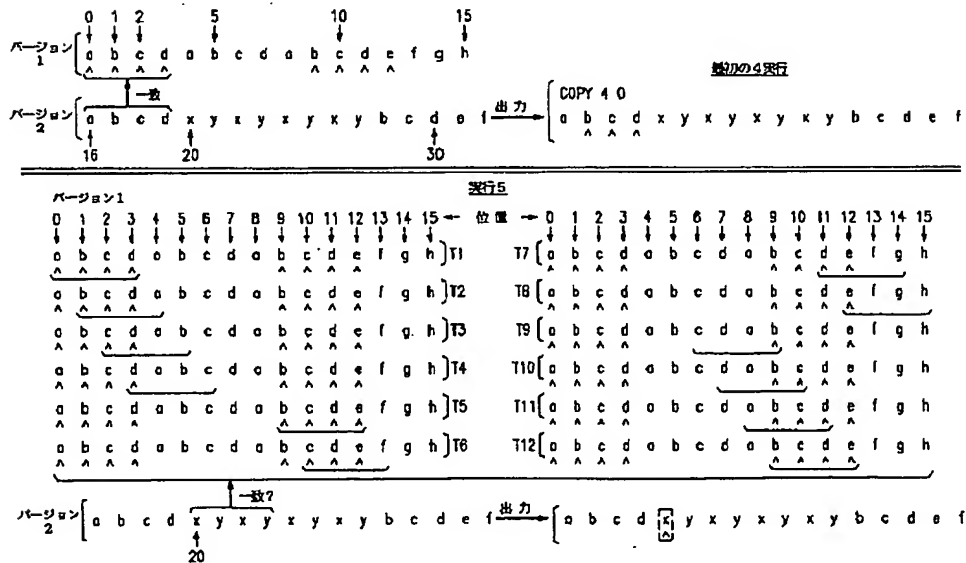
【図10】



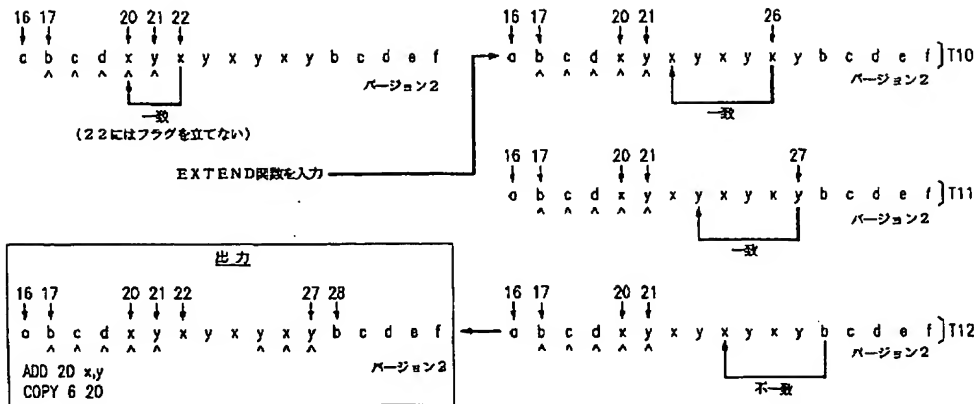
【図 2 F】



【図 2 G】



【図 2 H】



【図 3】

```

1. n = length(VERSION1);
2. C = 0;
3. while(true)
4. { inst = readinst();
5.   if(inst == end-of-file)
6.     return;
7.   s = readsize();
8.   if(inst == ADD)
9.     readdata(VERSION2+c,s);
10.  else
11.  { p = readpos();
12.    if(p < n)
13.      copy(VERSION2+c,VERSION1+p,s);
14.    else copy(VERSION2+c,VERSION2+p-n,s);
15.  }
16.  c = c + s;
17. }

```

【図 5】

```

1. Initialize table of positions T to empty;
2. process(VERSION1);
3. process(VERSION2);
4. process(seq);
5. { n = length(VERSION1);
6.   m = length(seq);
7.   c = 0;
8.   add = -1;
9.   pos = -1;
10.  len = MIN-1;
11.  while(1)
12.  { while(1)
13.    { p = search(T,seq,c,len);
14.      if(p < 0)
15.        break;
16.      len = extend(T,seq,c,p,len);
17.      pos = p;
18.    }
19.    if(pos < 0)
20.    { if(add < 0)
21.      { add = c;
22.        if(seq == VERSION1)
23.          insert(T,seq,c,0);
24.        else insert(T,seq,c,n);
25.        c = c + 1;
26.      }
27.    }
28.    else
29.    { if(seq == VERSION2)
30.      { writeinst(add,c,pos,len);
31.        p = c+len-(MIN-1);
32.        while(p < c+len)
33.        { if(seq == VERSION1)
34.          { insert(T,seq,p,0);
35.            else insert(T,seq,p,n);
36.          }
37.          p = p+1;
38.        }
39.        c = c + len;
40.        add = -1; pos = -1; len = MIN-1;
41.      }
42.    }
43.    if(c >= m-MIN)
44.      break;
45.  }
46.  if(seq == VERSION2 and add >= 0)
47.    writeinst(add,m,-1,0);

```

【図 8】

```

1. extend(T,seq,c,p,len)
2. { n = length(VERSION1);
3.   if(p < n)
4.     str = VERSION1;
5.   else
6.     str = VERSION2;
7.   p = p-n;
8.   m = length(seq); i = c + len + 1;
9.   n = length(str); j = p + len + 1;
10.  while(i < m and j < n)
11.  { if(seq[i] != seq[j])
12.    { break;
13.    }
14.  }
15.  return i-c;

```

【図 7】

```

1. search(T, seq, c, len)
2. {
3.   n = length(VERSION1);
4.   p = c + len - (MIN-1);
5.   key = seq[p, p+1, ..., p+MIN-1];
6.   for(each entry e in T that matches key)
7.   {
8.     pos = position(e);
9.     if(p >= n)
10.    {
11.      str = VERSION2;
12.      q = pos - n;
13.    }
14.    else
15.    {
16.      str = VERSION1;
17.      q = pos
18.    }
19.    d = q - (len - (MIN-1));
20.    if(d >= 0)
21.    {
22.      if(seq[c, c+1, ..., p-1] == str[d, d+1, ..., q-1])
23.        return pos - (len - (MIN-1));
24.    }
25.  }
26.  return -1;
27. }

```

フロントページの続き

(72) 発明者 カームーフォン ヴォー
 アメリカ合衆国 07922 ニュージャージー
 イ、 パークレイ ハイツ、 スウェンソン
 サークル 80

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 08-255104

(43)Date of publication of application : 01.10.1996

(51)Int.Cl.

G06F 12/00

G06F 12/00

G06F 9/06

(21)Application number : 07-325290

(71)Applicant : AT & T CORP

(22)Date of filing : 14.12.1995

(72)Inventor : KORN DAVID GERARD
KIEM-PHONG VO

(30)Priority

Priority number : 94 355889 Priority date : 14.12.1994 Priority country : US

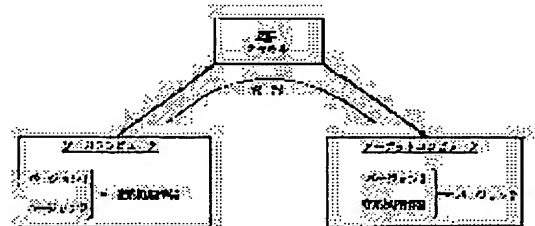
(54) EFFICIENT AND HIGHLY SAFE UPDATING OF SOFTWARE AND DATA

(57)Abstract:

PROBLEM TO BE SOLVED: To provide a method for reducing a transmission data amount, preventing interception and updating a software or data from a remote location.

SOLUTION: Relating to a device for updating a data file from an old version to a new version, the old version and the new version are compared and deformation processing information including information relating to similarity and difference is led out. Further, in order to lead out the new version, without referring to the new version itself, the deformation processing information is used and the old version is processed. Even without transmitting an updated file itself, by transmitting the deformation processing information to plural places, the plural versions of an original file such as the records of banks present at the plural places are updated.

Generally, since the entire contents of the new version are not included in the deformation processing information, this method is provided with high resistance to the interception.



THIS PAGE BLANK (USPTO)

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ **BLACK BORDERS**
- ☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- ☐ **FADED TEXT OR DRAWING**
- ☐ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- ☐ **SKEWED/SLANTED IMAGES**
- ☐ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- ☐ **GRAY SCALE DOCUMENTS**
- ☒ **LINES OR MARKS ON ORIGINAL DOCUMENT**
- ☐ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- ☐ **OTHER: _____**

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.

THIS PAGE BLANK (USPTO)